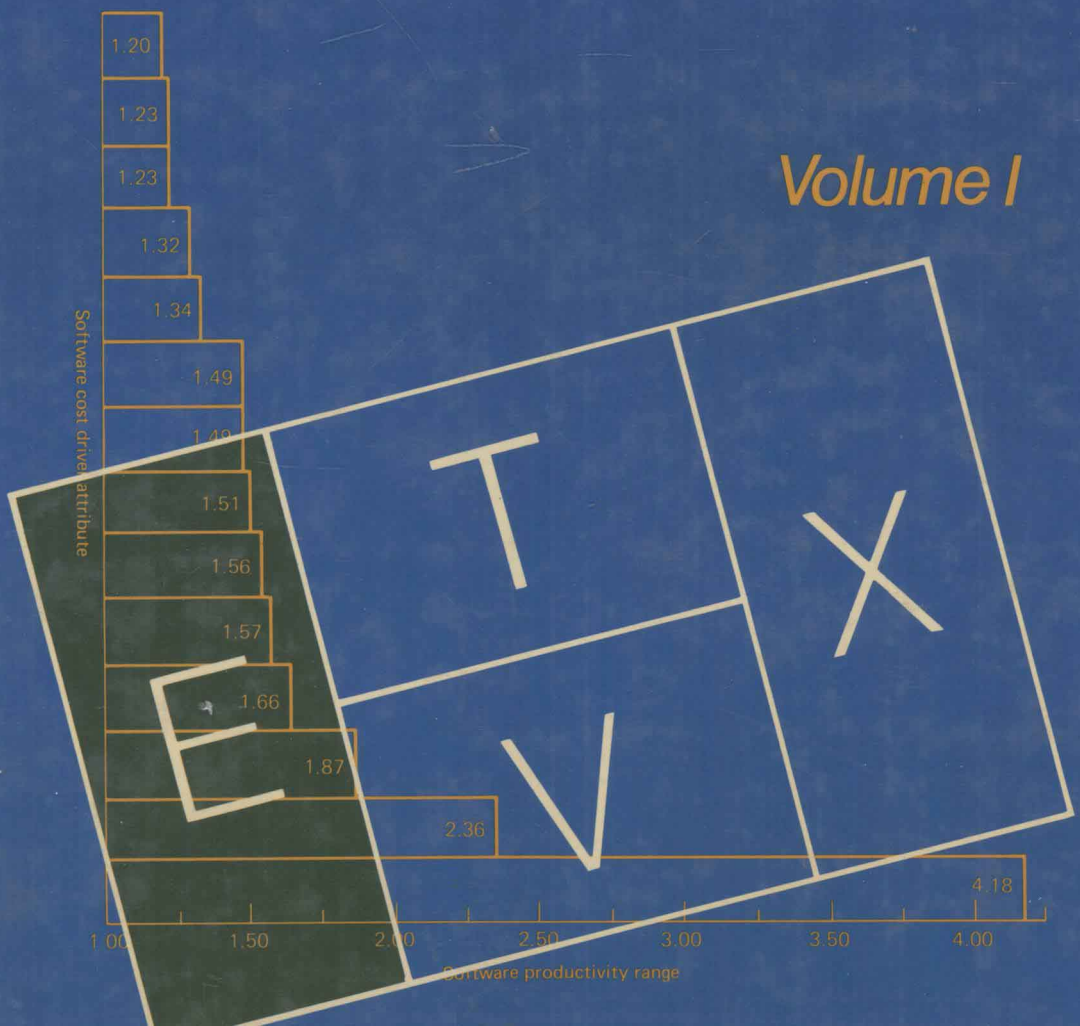
RONALD A. RADICE
RICHARD W. PHILLIPS

SOFTWARE ENGINEERING

AN INDUSTRIAL APPROACH

Volume I



Software Engineering: An Industrial Approach

Volume 1

Ronald A. Radice and Richard W. Phillips

Rensselaer Polytechnic Institute
and
International Business Machines Corporation

Prentice Hall, Englewood Cliffs, New Jersey 07632

Radice, Ronald A. (date)
Software engineering.

Bibliography: v. 1, p.
Includes index.
1. computer software—Development. I. Phillips, Richard W. II. Title.
QA76.76.D47R327 1988 005.1 87-30037
ISBN 0-13-823220-2 (v. 1)

Editorial/production supervision: *Gloria L. Jordan*
Cover design: *Photo Plus Art*
Manufacturing buyers: *R. Washburn, C. Grant*

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, N.J. 07632

© 1988 by Prentice-Hall, Inc.
A division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

Ada is a registered trademark of the United States Government (Ada Joint Program Office)
LOTUS is a registered trademark of Lotus Development Corporation
PSL/PSA is a registered trademark of ISDOS, Inc.
SADT is a registered trademark of SofTech, Inc.
SREM is a registered trademark of TRW Corporation
VisiCalc is a registered trademark of VisiCorp

Reproduced by Prentice-Hall from camera-ready copy prepared by the authors using the IBM Publishing Systems BookMaster ™.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-823220-2

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*

Dedicated to Claire and Lee

Foreword

Software Engineering is only a human generation old. At that age, civil engineering had yet to discover the right triangle. While more people are working on software engineering now than worked on civil engineering then, fundamental ideas and principles still take time to emerge. For example, it took the genius of Edsger Dijkstra several years to articulate the idea of structured programming after the advent of ALGOL 60.

Even before the discovery of structured programming, IBM undertook the largest, most complex software development project of its time, OS/360, under the leadership of Frederick Brooks. Although software systems have seldom had lifetimes of more than ten years (with changing hardware generations), OS/360 has been evolved and extended over twenty years, and no horizon is yet in sight. With requirements of hundreds of thousands of installations accepting ever new generations of hardware and operational conditions of distributed real time data processing, the methods and tools used in this evolution have been subjected to an uncompromising, relentless proving ground without precedent.

The demands of all this massive and continuing software development have led to a new understanding and crucial perspective that embeds methods and tools into a framework of intellectual and management control. This framework is called the IBM Programming Process Architecture, and it is the central and unifying theme in this book. As the authors state:

> The Programming Process Architecture is the highest representation of the software process. While it includes the concept of a software engineering environment, it contains much more in a broader framework.

In short, it is an industrial strength process that has been developed and used in large scale practice with consistent results and control.

此为试读,需要完整PDF请访问: www.ertongbook.com

Two principles learned from large scale software development are brought forward in this book: (1) rigorous definition and management of the process beyond the levels exerted by individual tools and techniques are essential, and (2) the evolving process, as architected and practiced, must be allowed to govern the requirements for new tools.

In accordance with the scope of software development supported by the IBM Programming Process Architecture, this book provides a broad coverage of software methodologies, from formal procedures to creative heuristics, with many references to original work in the field. Classroom-tested in university courses, this book represents a first in bringing the lessons of megascale software development from the industrial crucible to the university classroom.

Harlan D. Mills

Preface

Why Another Book on Software Engineering?

There are many excellent books already available, and we have used a number of them over the years in the course we have taught at Rensselaer Polytechnic Institute (RPI). While we were developing our two-semester graduate-level course sequence in Software Engineering, we realized that none of the existing books fully met our needs for the sequence we were teaching, as none specifically represented an industrial view. The industrial view we wanted to focus upon required a book that was: (1) based on industrial experience and examples and (2) emphasized the engineering of software. We therefore found ourselves developing volumes of course notes and handouts to document our lectures for our students. The course notes for the first semester have now completed their migration into this book. This volume represents the first in a proposed series which will address the complete life cycle of programming development. The first two volumes will cover the life cycle of software development, and it is intended that succeeding volumes will cover in more detail each of the stages of development. Emphasis in this volume is placed on the methodologies within each development stage, and for each stage we note our recommended approach. This volume directly maps to the first semester as we teach it at RPI.

Enough has been written over the last twenty years about the software crisis to cause one to wonder if it will ever be solved. Clearly, since it was first uttered in 1968 at the North American Treaty Organization (NATO) conference in Garmisch, Germany, the problem has changed somewhat. Indeed much has gotten better and much will continue to further improve in the coming years. The combined and increasing focus of industry and academia on this problem will bring the evolution of Software Engineering to its required maturity in the industry and engineering disciplines. We will explore some of the historical aspects of this crisis in Chapter One. However, we do not intend to belabor this crisis in this book, but will, by example, speak more optimistically about how it can best be addressed today and what the near future holds as opportunity.

Any technical book can soon become out of date, especially in a field that is evolving as quickly as Software Engineering. Nonetheless, we believe the basic underlying principles that we highlight are necessary to both short-term and long-term Software Engineering solutions. In fact, these basic principles are evident in many of the successful project solutions we have seen. Some of the specifics we discuss in the book are undoubtedly evolving even as we go to print, but we believe the basics will persist.

It is essential that computer science and software engineering students get an industrial view in their education. It is important to reiterate here that the aca-

demic world must do more to introduce and address the realities and the problems of industry in the Computer Science/Software Engineering curricula. It is important at this time, given the maturity level of software engineering, that the industry and academia be more widely integrated. This can be accomplished by having academia draw people from industry to teach graduate and undergraduate programs, or by having universities assign members of their teaching staffs to industrial sabbaticals.

The alternatives within Software Engineering that we have chosen as best or preferred are naturally influenced by our experiences at the International Business Machines Corporation (IBM), but we have, in almost all cases, gone beyond our immediate experiences to confirm the effectiveness of our preferences or recommendations. Inside IBM, our focus is predominantly determined from personal knowledge in System/370 software product development. This history spans a combined 55 years in the industry. Our backgrounds cover a multitude of products from operating systems, to telecommunications, to languages and applications. It includes work across all stages of the programming development life cycle. Since 1972, we have been involved in defining, implementing, and changing Software Engineering within IBM. The views and opinions we hold in this book, although influenced by our work at IBM, are ours alone and not necessarily those of IBM.

Is This Book for You?

This book was written first with our students of Software Engineering at RPI in mind. As we developed the book, we realized that it applied as well to all software workers in the industry, whether they are students, new programmers, experienced programmers, or managers of programmers; and whether their focus is systems, embedded, or application programming. The book is appropriate to both academia and industry, and, therefore, we believe we have met our original objective when we were first asked by RPI to bring an industrial view of Software Engineering into the academic environment.

What Is This Book About?

The primary focus is *software process*: what it is, what is meant by it, why it is important, what its underlying principles are, and how it is managed. The result of a good process, in general, is a good product. There are exceptions on both sides of this relationship, but we believe that a good process is required to achieve good products repeatedly. There are other important aspects in the software environment, such as tools and methodologies, but we believe that the process should determine the tools and the methodologies and not vice versa. Throughout the development life cycle any particular process is only as good as the people who work with it and believe in it. Without good programmers and

managers behind it, no process by itself is sufficient to achieve the goals of better quality, higher productivity, and shorter development cycles.

In this volume we describe the project development stages through to the code stage. In each stage we emphasize the process paradigm we recommend in Chapter Two. Again in each of these stages we focus on an industrial approach.

A Quick Outline

Chapter 1 introduces the topic of Software Engineering: its purpose, problems, and evolution.

In Chapter 2 we present the dominant theme of the book, the software process. We discuss

1. Evolution and prevalent views of software process
2. Principles relevant to software process management
3. A recommended approach for a process definition and management

In Chapter 3 we explore how the software product should be planned. This product-planning activity is central to the first portion of the Requirements and Planning stage. It is a precursor to determining the feasibility of continuing the product's actual development.

In Chapter 4 we discuss the need for alternative methods and a recommended approach to Requirements Engineering.

Chapter 5 explores a number of ideas about human factors in software, including the idea that human factors should be treated as any other functional requirement for the product, but that specific process focus must be brought to bear in order to achieve it.

Chapter 6 is focused on the planning of the project, that is, what is the best way to plan, execute, and control the project process to deliver the product.

Chapter 7 introduces the design of software and relevant concepts necessary to engineer the design.

Chapter 8 discusses approaches for validating and verifying the completion of the various work items that are developed during the product cycle. We recommend preferred approaches.

Chapters 9 and 10 discuss relevant concepts, methodologies, and representations for completing software design at three discrete design levels: Product Level Design, Component Level Design, and Module Level Design.

Chapter 11 completes this book and addresses the Code stage of software development.

Subsequent stages of the software development process will be discussed in Volume 2 of this textbook series.

How to Use This Book

Our lectures follow the text, and are coupled with three other activities:

1. Course projects
2. Case studies
3. Additional readings

We follow a 14-week sequence and try to address the chapters in the following sequence:

Chapter 1: The State of Software Engineering — Week One

Chapter 2: The Process of Software Production — Week Two

Chapter 3: Planning the Product — Week Two

Chapter 4: Requirements Engineering — Weeks Three and Four

Chapter 5: Human Factors and Usability — Week Five

Chapter 6: Planning the Project — Weeks Five and Six

Chapter 7: Design of Software — Weeks Six and Seven

Chapter 8: Validation and Verification — Week Seven

Chapter 9: Product Level Design and Component Level Design — Weeks Eight and Nine

Chapter 10: Module Level Design — Weeks Ten and Eleven

Chapter 11: Code — Week Twelve

The course projects are designed to support Software Engineering. Students work on projects that address requirements formalisms, project history repositories, project tracking, design language processing, code restructure engines, reverse engineering, code generation, test coverage, test case generation, and software metrics, among others. Projects are assigned to directly reinforce the software engineering principles we teach in our lectures.

The case studies are intended to simulate specific types of problems that the software engineer might encounter in industry, and to structure a level of project progress during the semester. While we are hesitant to do anything to subvert the software principles we teach, we do have to contend with the artificial environment of a 14-week course. The focus during this semester, therefore, is in creating a learning environment more than in requiring a fully functional project at the end of Semester One. Nonetheless, the project teams are required to demonstrate the capability of their "product" at a level of test completeness by the end of the semester. Usually the teams achieve completion at a Unit Test level, a test effort which immediately follows the completion of the code. In some cases testing has gone well beyond the Unit Test level. Basically we are simulating a product development cycle for the initial release of a product within a 14-week constrained period. During the second semester the projects are rotated, the teams are kept together as much as is practical, and the teams are asked to continue with

1. Completing Release 1
2. Using it throughout the rest of the semester
3. Maintaining their "product"
4. Developing the requirements for Release 2
5. Completing Release 2 through the System Tested level in the life cycle

Finally, additional readings are assigned to either stress major points we want the students to explore or to take advantage of current findings in the software engineering literature.

A student using this book can either do so directly in a course that uses the text or read it as stand-alone text. While it is anticipated that the text be used in a Software Engineering course, we wrote it so that it will also stand by itself.

Acknowledgments

As with all books of this nature, there are more people who deserve recognition than we can adequately acknowledge in our words here. First, we must thank our secretaries, Gloria Murphy and Anne Wendt, who helped us in so many ways with typing, editing, copying, making the graphics, and the numerous last-minute requests for changes. Fortunately, they had a well-defined process and good text processing tools to keep us under control and to help us make this text. We thank Ron Wendt for showing us the powers and intricacies of Publishing Systems BookMaster, the product we used to produce this book in camera-ready form. Next, we have to thank the many past students at RPI who were in our

此为试读,需要完整PDF请访问: www.ertongbook.com

classes and worked with our notes as they evolved towards this book. We also thank Herb Freeman, who initially contacted us for RPI. We want to thank IBM for giving us first the experience and then the support while we worked on this book. Finally we thank all of the reviewers who helped make this a better book than it otherwise would have been.

R. A. Radice and R. W. Phillips
Poughkeepsie, New York

Reviewers

Joan Carl
Gerhard Chroust
Robert Goldberg
Jack T. Harding
Gene F. Hoffnagle
Ev Merritt
Harlan D. Mills
Robert P. Mueller
Almerin C. O'Hara, Jr.
Leonard Orzech
W. C. Peterson
A. M. Pietrasanta
Bernie Rachmales
Claire L. Radice
Hans A. Schmid
Carol A. Schneier

Contents

此为试读,需要完整PDF请访问: www.ertongbook.com