THE SOFTWARE FACTORY

A Fourth Generation Software Engineering Environment

Michael W. Evans

WILEY

A WILEY-INTERSCIENCE PUBLICATION

JOHN WILEY & SONS

New York • Chichester • Brisbane • Toronto • Singapore

Library of Congress Cataloging-in-Publication Data
Evans, Michael W.
The software factory: a fourth generation software engineering environment
Michael W. Evans.
p. cm.
"A Wiley-Interscience publication."
Bibliography: p.
Includes index.
ISBN 0-471-01192-4
1. Software engineering. I. Title.
QA76.758.E98 1988
005.1—dc19 88-10819
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

THE SOFTWARE FACTORY

A Fourth Generation Software Engineering Environment

To Victor and Irene Rome; Parents and Friends

Foreword

The software manager of a large project is faced with a bewildering set of choices in establishing an environment. Competing and contradictory claims emanate from a variety of sources, including vendors, technical literature, the user for whom the project is being developed, different factions within the project development team, and so on. There is no single product, book, or managerial style that will lead magically to a project development environment in which all parties are automatically made aware of the myriad of details that go into day-to-day project decision making. Such details include, but are by no means limited to, current project status, schedule, technical products, managerial decisions, quality assurance pronouncements, change requirements, staffing levels, hardware and software configurations, and test results.

In the absence of a single product panacea that would make life simpler for the project manager, Michael Evans has produced a book that gives the manager the perspective, information, and courage to formulate a cohesive, uniform project structure, supported by a workable software environment, in which different parts of the team work toward common goals. This book also allows the manager to deal intelligently with vendors, in terms of deciding which software tools would enhance productivity, and fit within the framework of the environment.

The book has utility far beyond the project manager. Students studying computer science or business, for example, may wish to know what the workaday situation is like in the business and industrial world, and in particular, how this world differs from academia, and how he or she may

make the transition from student to professional software engineer with a running start. This book contains a wealth of information that answers such concerns. Therefore, this book could be of value as a text in college courses on data processing, whether taught in a business department, computer science department, or an economics department.

Likewise, customers of major projects, who may have a great deal of approval authority over the environment, will benefit from this book. They will have a focused way of evaluating proposals, be able to judge the quality of the winning management team, and have a forceful, well-thought-out perspective on major environmental issues.

LARRY YELOWITZ

Preface

Programming is approaching the software factory concept, when program modules are the engines, tires, and transmissions produced; design documents are the blueprints; and operational documents are the shop repair manuals. In the software factory, analysts design while programmers manufacture and repair software systems. Analysts no longer create—they manufacture, and quality control is an important part of that process.

This optimistic view of the state of the software industry was presented by Jay Arthur in his excellent book *Programmer Productivity* (1983). Unfortunately, the reality is that software development is still, in large part, a technical art form.

The promise of an integrated environment for the development and support of software approaching an assembly line has proven to be an elusive goal. The software environment was first envisioned by Jack Munson while at Systems Development Corporation during the late 1960s. Subsequent work by Thompson Ramo, Woolbridge and IBM advanced this concept, yet the promise of more predictability in the software development workplace was not fulfilled (Brateman, 1975). Even the Japanese, who have invested heavily in the development of this concept, have yet to fully realize the benefits (Wasserman, 1981).

The software engineering environment is more than just a suite of tools. It is an integration of methods, data products, development practices, life-cycle and documentation requirements, assurance practices, business and management requirements, and automated support. These compo-

nent parts, when applied to a software project, ensure a smooth and complete development or support framework.

This book describes the fourth-generation software engineering environment and demonstrates why the concept of integration is so difficult to implement. This concept is a vision of the future. It has not been fully realized despite the many descriptions of "integrated environments" and successful experiences using automated software methods that are presented in the literature. Application of the development methods described in this book will provide an environment in which quality software can be produced in a predictable, controlled, and productive fashion. The approach recognizes the need to integrate the various elements of the software development process with software management and control procedures. The environment described is disciplined and provides the means by which quality, productivity, and product acceptability will be engineered into the software life cycle.

WHY DO WE NEED THE SOFTWARE ENGINEERING ENVIRONMENT?

This book describes the components that must be considered when defining, configuring, and applying the software environment. The software engineering components presented are conceptual; they apply to software engineering projects in general, irrespective of the approach used to support the project. The software environment presented in this book is a means of organizing the concepts into a cohesive and organized structure for software development and support. It is not strictly a toolset, a method for development, or a structure of data management or control.

The need to meet the software challenge successfully touches every major business and government entity. According to Barry W. Boehm in his book *Software Engineering Economics* (1981), during the mid-1980s more than 40 million workers—40% of the U.S. labor force—depended on computers to some degree in their work.

This proliferation of computers has brought with it a disturbing realization: Unless the software development process becomes more productive and the software products more predictable, the computer revolution will be geared to the pace of software development. This situation is unacceptable. Our software development resources must be made more productive.

Many unique, innovative, and often esoteric software development techniques are being developed by industry, government, and academia to meet the software challenge. Each of these is attempting to fulfill the elusive dream of increased productivity while improving the quality of software products and services. Each of these techniques centers on a single or limited set of activities that take place during development. Integration of these techniques into a cohesive software development environment has been neglected.

The lack of software project integration has resulted in frequent, and often dramatic, shortcomings in software technology. These problems too often reduce the effectiveness of the software project. They restrict the benefits of software innovation and limit the application of technology to actual development situations.

Can we afford this continued software technology shortfall? Barry Boehm states that the annual industry and government software expenditure was $40 billion, or 2% of the gross national product (GNP) in 1984. The need for software development resources is increasing much faster than the general economy; it now represents the bulk of the computer and information-processing industry. This industry is projected by Boehm (1981) to be 8.5% of the GNP by the mid-1980's, growing to 13% by 1990.

Concurrent with this rise in demand has been a dramatic decline in hardware costs. Increased demand coupled with declining costs has resulted in general and increasing use of computers throughout society, not just in the technical disciplines. This dependence on computer products indicates an increasing dependence on software.

The spiraling demand for computers has focused attention on the factors that inhibit software development: inadequate methods, and poorly defined, imperfectly integrated software development practices and tools. These problems are compounded by poorly defined or rapidly changing user requirements. In these areas, the software industry is like the cobbler's children.

The basic software development resource is labor. Despite the extensive demand for software, there is and will continue to be a critical shortage of software professionals—those who plan, develop, test, and support software.

According to the Department of Defense, there are 50,000 to 100,000 fewer qualified software professionals than are required to support current industry needs adequately. Projecting this shortfall against the increasing demand for software, by the end of this decade the shortage is predicted to increase to between 860,000 and 1 million software professionals. The only alternative to reducing our expectations because of personnel shortages is to enhance the productivity of those that we employ (Boehm, 1974).

To date, our efforts to meet this challenge have been relatively unsuccessful. Increasing software development productivity and product quality requires improved use of computers. Such optimization of computers, in turn, depends on the availability of adequate software. Poor software tools and a lack of common understanding of the components and interactions that exist in a software project make it impossible to solve development challenge through automated means.

In the private sector, when we fail to address the software problem adequately, product and service quality and customer responsiveness suffer, the bottom line is affected, and the reputation of the company invariably suffers. Government impacts may be even more significant. Software

development problems may affect national defense, jeopardize human life or safety, reduce national preparedness, and cause delays in regulatory actions. Taken to an extreme, the effects of poor software can threaten our national existence if one considers our reliance on computer systems.

Solving the software challenge requires that we treat the many disciplines associated with development as an integrated engineering strategy, not a technical art form. Using the data products that are produced during the development process, this strategy must tie the software planning, development, and support activities together. The software environment must integrate the use of disciplined methods and procedures supported by appropriate tools. This approach acknowledges that software development is far more than the writing of computer programs. It is the recognition, tailoring, integration, and application of a variety of disciplines to a consistent life-cycle approach to software development. It must take into account the analysis and specification of requirements, design, programming, testing, integration, and, finally, support of the software product. This is the core of fourth-generation software engineering technology.

THE COMPLEXITY OF THE SOFTWARE ENVIRONMENT

This book describes the issues to be raised, the questions to be asked, and the problems to be addressed when establishing a consistent and reproducible fourth-generation software development environment. It describes how to link methods together, how to plan the development and control of data flow and the application of technology, and how to address the issues that affect development productivity. It is intended for software engineers and project personnel who are concerned with the integrity of the products and processes that are part of software development.

The concepts presented are theoretical but are not beyond the current state of the art or practice. The environment described in this book is based on the rigorous application of accepted development methods that can be applied to small projects, large development or support projects, and a variety of projects with different technical requirements. The concepts must be adapted to specific project situations. The vision of the factory environment presented here does not represent a universal solution to the problems plaguing the software industry. These can be solved only by understanding the specific, unique problems facing each project and making a commitment to address them in a consistent and focused manner.

Morgan Hill, California
February 1988

MICHAEL W. EVANS

REFERENCES

Arthur, Lowell J., *Programmer Productivity: Myths, Methods and Murphology*. New York: John Wiley & Sons, 1983, p. 127.

Bratman, H., The Software Factory, *Computer Magazine*, Vol. 8, No. 5, 1975, pp. 28–35.

Boehm, Barry, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

Boehm, Barry, et al., CCIP85. Washington, D.C.: Department of Defense, 1974.

Wasserman, Anthony, Toward Integrated Software Development Environments. Tutorial: Software Development Environments. *IEEE Computer Society, Compsac*, 1981, p. 16.

Acknowledgments

This book is a reflection of the author's experience. Much of this experience is a result of the author's work with NASA over the past 4 years. The information system life cycle and verification approaches described are a direct result of this work. I would like to thank Bill Wilson and Don Sova of NASA headquarters, as well as Sue Voigt, Judy Steinbacher, Dr. E. D. Callender, and the many others whose creative ideas have been used in this book. This experience has proved invaluable and, hopefully, is reflected in the chapters that follow.

This book would not have been possible without the help and support of my wife, Charlotte, who spent hours sprucing up the manuscript, and reworked certain pages repeatedly to make this book possible. I would like to acknowledge the many persons who helped in the production and review of the manuscript: Barbara Zirolli of Expertware, whose expertise in configuration management was freely offered and is incorporated in the text; Jack Bond of the Department of Defense; Candyse Barr of ESL Inc., whose review of the text and inputs on life cycle management were essential and greatly appreciated; and Gary Furr, again of Expertware, whose early editing got me off on the right foot.

Jack Garman of NASA and Priscilla Albright reviewed the manuscript. Their comments, although painful, were valuable and essential.

Dr. Larry Yelowitz of the Ford Aerospace Corporation provided technical input regarding descriptions of methods and was a constant source of encouragement.

I would like to offer special thanks to Barbara Christoph, whose encouragement and review were critical to the book's development. Most impor-

tantly, I would like to thank E. D. Callender, who had the courage to tell me that the early manuscript was no better at 31,000 feet than it was on the ground. I hope that I have addressed his concerns.

Finally, I would like to thank Maria Taylor, my editor at John Wiley & Sons, for the encouragement and support that made the book possible.

M.W.E.

Contents