Software Engineering

Merlin Dorfman
Richard H. Thayer

Preface

This tutorial describes the current state of the practice of software engineering. The purpose for writing this tutorial is twofold:

1. There is a need for a set of papers that can be used for a senior or graduate class in software engineering for those faculty members who prefer to use a set of definitive papers rather than a textbook, while relieving the instructor of the need to obtain copyright clearance from dozens of publishers.
2. There are software professionals who would like to have a preselected volume of the best papers in the field of software engineering, for self-study or for a training course in industry.

For the purposes of this tutorial, software engineering is defined as an engineering discipline that applies sound scientific, mathematical, management, and engineering principles to the successful building of large computer programs (software).

> **software engineering**. (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software. (2) The study of approaches as in (1).

Software systems are built within an organizational structure called a software project. A *successful* software project delivers its planned products within schedule and budget and meets its defined functional and quality requirements.

Software engineering includes software requirements analysis; software design; modern programming methods; testing procedures; verification and validation; software configuration management; software quality assurance; tools for analysis and design; corporate software policies, strategies and goals; project management planning, organizing, staffing, directing, and controlling; as well as the foundations of computer science.

In our opinion, good tutorial papers have the following characteristics:

- They define the basic terms
- They cover the state of the practice for the given topic thoroughly and evenly
- They avoid new, unproved concepts (other than to list them as future possibilities)
- They do not try to sell one tool or concept over all others
- They are easy to read
- They are organized in a hierarchical manner (top-level concepts discussed first, second level concepts discussed next, and so forth)
- They provide additional references
- They come from a refereed journal (unless written specifically for the tutorial)
- They were written by an expert in the area (to assure all of the above)

Our criteria are of course idealized. Even these "rules" can be violated if there is a good reason.

In addition, to keep the whole tutorial under 500 pages, each article should be no longer than 10–12 journal pages.

Our intent was to use the best and most current leading papers in the field. To assure that our intent was fulfilled, we sent survey forms to over 200 of the leading researchers and practitioners of software engineering in the US, Canada, Europe, and Asia, asking what papers they would like to see in a software engineering tutorial. Seventy survey forms were returned. In a surprisingly large number of the basic specialty areas of software engineering, there were no recent, high-quality overview papers identified. It appears that, as a discipline matures, people no longer write overview papers. There have been very few acceptable papers on the fundamentals of software engineering published in the last ten or so years. Therefore the editors contacted some of the leading authors and practitioners in the major subfields of software engineering and asked them to write papers for us. A list of the contributors can be found on Page v; we are grate-

ful that they took time from their busy schedules to write for this tutorial.

Our tutorial is divided into four parts and 13 chapters.

Part One includes Chapters 1, 2, and 3 and provides an overview of software engineering in the context of current issues and the engineering of large complex systems. Chapter 1 describes the problems that occur in developing software, sometimes called the "software crisis." Chapters 2 and 3 present the concepts of system engineering of software-intensive systems, and of engineering of software products, as the solution to the "software crisis."

Part Two, Chapters 4 through 8, describes software engineering from the viewpoint of the phases of the software life cycle: requirements, design, implementation (coding), testing, and maintenance. Chapter 4, *Software Requirements Engineering and Software Design*, discusses the state of the practice in requirements and design. Originally requirements engineering and design were separate chapters, but most papers on the subject combine the two topics so we did as well. Chapter 5, *Software Development Methodologies*, also combines approaches supporting analysis and design. Because of their growing importance, special attention was paid to object-oriented and formal methods. Chapter 6, *Coding*, describes programming activities as they affect software engineering and vice versa. Chapter 7, *Software Validation, Verification, and Testing*, and Chapter 8, *Software Maintenance*, describe the state of the practice in those areas of specialization.

Part Three consists of Chapters 9, 10, and 11 and takes a phase-independent view of the software development process and its management. Chapter 9 looks at software quality assurance in the larger context of ensuring conformance to the development process, as well as in the traditional smaller-scale context. The chapter also looks at configuration management, standards, and reliability engineering as keys to building quality into software products. Chapter 10 discusses software project management and some related topics such as software cost estimation and risk management. Chapter 11 looks at the software development process and how it fits into the larger scope of the software life cycle.

Part Four discusses software technology and education. Chapter 12, *Software Technology*, discusses how technology is transitioned from theory to practice as well as software re-engineering and reuse, computer-aided software engineering (CASE), and software metrics. Chapter 13 contains a single paper on the topic of education for software professionals.

This tutorial is a companion document to the below-listed software engineering tutorials. Duplication of papers has been kept to a minimum. In a few cases, particularly important papers are duplicated in order that each tutorial can stand alone.

- R.H. Thayer, editor, *Tutorial: Software Engineering Project Management,* IEEE Computer Society Press, Los Alamitos, Calif., 1988 (revision in process).
- R.H. Thayer and M. Dorfman (eds.), *System and Software Requirements Engineering,* IEEE Computer Society Press, Los Alamitos, Calif., 1990 (revision in process).
- R.H. Thayer and A. D. McGettrick (eds.), *Software Engineering—A European Perspective,* IEEE Computer Society Press, Los Alamitos, Calif., 1993.

We would like to acknowledge the support provided by the following people.

- Ms Catherine Harris and Dr. William Sanders, managing editors of IEEE Computer Society Press
- IEEE volunteer editors under the direction of Prof. Jon Butler of the Naval Postgraduate School
- Fernando Proaño, a graduate student at Sacramento State University, who assisted us with the survey mentioned above

Merlin Dorfman, Ph.D.
Lockheed Martin Missiles and Space Company
Sunnyvale, California, USA

Richard H. Thayer, Ph.D.
California State University, Sacramento
Sacramento, California, USA