Japanese Perspectives in Software Engineering

Edited by

Yoshihiro Matsumoto

Yutaka Ohno

Japanese Perspectives in Software Engineering

Edited by

Yoshihiro Matsumoto
Kyoto University

Yutaka Ohno
Advanced Software Technology and Mechatronics Research, Institute of Kyoto

ADDISON-WESLEY PUBLISHING COMPANY

Singapore • Wokingham, England • Reading, Massachusetts
Menlo Park, California • New York • Don Mills, Ontario
Amsterdam • Sydney • Bonn • Tokyo • Madrid • San Juan

Foreword

Software engineering, as a distinct subject of research, study, and application, is 20 years old this year. After a sometimes tumultuous infancy, it is coming of age in a number of different ways throughout the world. This excellent collection of articles illustrates that point well and, in the process, emphasizes the highly interdependent, international character of modern science and engineering.

What you will find here is a set of articles that appear to represent a solid cross-section of current work in software engineering at that interesting point of intellectual development when ideas emerge from the laboratory and start to find their way into the daily work of industrial organizations. Some articles describe 'prototype' explorations of new software development technologies; others outline the usage of the current stock of operational development technologies in Japanese organizations; and others sketch possible future paths of Japanese research and development on, as well as application of, software technology. All are written from a Japanese perspective, and all provided me with a view in one way or another of what is going on in Japan today.

I should note that it is not my place to 'certify' the views put forth here, nor could I even if that were asked of me. I do know, through a number of years of interaction with Japanese researchers and industrial practitioners in various ways, that the perspectives given here (both explicitly and implicitly) are very consistent with what I and others have independently observed.

I will, however, venture a personal viewpoint. The articles here document what has been obvious from individual papers and visits for a number of years; namely, that the Japanese have studied carefully the software concepts and industrial practices of the West, choosing and adapting those that are most relevant to their needs, then integrating them into coherent systems for software production. (I would hasten to note that some of those early ideas were, in fact, developed in Japan and/or by Japanese researchers as well.)

What is new, and what constitutes one of the forms of maturation I mentioned above, is that today in the realm of utilization of the best of current software development technology and the creation of the next level of capability the Japanese are equal players on the international stage. Several of the articles here reflect an understanding of the issues and a pragmatic approach to pushing the frontiers of software technology forward that is equal to or more advanced than that of any in the world. As these perspectives illustrate, we can expect the Japanese to contribute their share of the good ideas and demonstrations of pragmatic capability for the next generation of software technology.

Science and engineering have long held to an 'internationalist' view that ideas and data recognized no political boundaries (at least under normal circumstances). Indeed, one measure of a new discipline is sometimes held to be the extent to which it is internationally pursued. While software engineering was certainly international in its birth, much of the intellectual development of the field has resulted from a relatively small number of people in the United States and Western Europe who mostly knew each other and communicated in person.

The papers here, better than any single collection that I have seen, illustrate that software engineering has started to mature along the lines of traditional scientific work. Ideas published elsewhere have been picked up, applied, refined, extended and built on by the Japanese (largely in an 'arm's length' mode). This has resulted in a viable research and development community in Japan that is now producing its own new ideas and publishing them for the rest of the world. This is as it should be.

As with any collection of work by a variety of authors, you will not get a completely uniform view, a consistent level of focus or the answer to all your problems (be they pragmatic or research questions). The editors do not intend this, and it is another aspect of the maturation of software engineering that most people today understand that such simplistic viewpoints are out of place. Instead, we understand that the development of software is a complex, intellectual task to which a variety of ideas, experiences and data apply. I recommend this collection to you and hope that you enjoy it and learn as much from it as I did.

Peter Freeman
University of California, Irvine

Preface

The total domestic demand for software by the Japanese public and industries in the future is predicted to rise 6% every year in monetary value. A lot of scientific and technical research has been conducted to increase the capability for producing reliable commercial software in order to catch up with these needs.

The first article, by Yutaka Ohno, addresses many of the basic problems related to the aforementioned requirement and summarizes many efforts both in academic and industrial Japanese organizations.

Part 1 organizes various new research accomplishments and practices which deal with requirements specification. The software development life cycle model is changing. Some of the new paradigms such as operational specification, prototyping, program transformation and automatic program generation are discussed.

Part 2 is a mixture of industrial practice and academic efforts. Computer-assisted software design, testing and quality assurance are discussed first and then some of the topics of the future, AI application and algebraic specification, are discussed in the second half.

Part 3 addresses practical problems, especially software engineering environments and management of software production. Several large Japanese software industries apply a software production system called the software factory. Discussions of the software factory are included.

The discussions of these themes will contribute in helping readers of this book to understand:

- the current interests of Japanese universities and industries in the pursuit of the problems addressed at the beginning of this Preface;
- the latest accomplishments of the research and development activities in Japan related to software engineering; and
- the real management practices applied in Japanese industries.

Contents

1

Background and Current View of Software Engineering

*Yutaka Ohno**

Abstract

The principal software engineering challenge, achieving higher development productivity and quality, is discussed in connection with the recent trends in the information technology field. The history and characteristics of software engineering are surveyed, showing past, currently effective and essential results of research.

Developments in software engineering are discussed, including the software life cycle model, software paradigms, use of knowledge engineering and natural language, automatic generation, software management and development environments. Finally, the kernel mechanism is proposed as a software development environment.

*Emeritus Professor, Kyoto University, and President, ASTEM Research Institute of Kyoto, Japan

1.1 The problem of software [1–3]

Computer hardware technology achieves remarkable progress year after year, and the cost : performance ratio of computer hardware has been improved tremendously, which promotes the use of computers in many ways within the fields of industry, the economy, administration and science. The infrastructure of the so-called information-intensive society is evolving gradually. To support this trend, it is the responsibility of software to deal thoroughly with various jobs or functions in every field or department. A more abrupt increase in demand for software is expected in the future. Furthermore, the demand for software which contributes to the information-intensive society will increase not only in quantity but in quality. As all sorts of information systems and control systems grow in size and complexity, and are distributed and networked, a higher degree of reliability is sought for software. Moreover, to obtain a more humanized system, we may expect higher intelligence from software. To provide a more friendly and cooperative system, greater sophistication for the man–machine interface is desired. A qualitative metamorphosis of software is necessary to meet these requirements (Table 1.1).

In fulfilling these requirements, software costs have come to account for more than 80% of the total costs of information systems, and the gap between the demand and supply of software has become wider and wider. Table 1.1 illustrates this situation with respect to the growth of the information-processing industry. In this table, we observe that the growth of programmer productivity is lower than the growth of the information-processing industry and the performance of computer hardware, and hence an urgent countermeasure is required. The Ministry of International Trade and Industry of Japan has recently estimated that the shortage of programmers may be 970 000 in AD 2000 if we do not take any countermeasures. In conclusion, the present problem of software is how to develop efficiently a software product with high quality and reliability.

Some people may recall, by analogy, a technique for manufacturing material products. Modern technology brought the manufacture of various types of products with high quality and high productivity into practice with factory automation rather than with human skill. The

Table 1.1 *Recent trends of information technology fields*

Information technology fields	Growth magnification per 10 years
Industry	4
Machine performance	10^2
Programmer productivity	2.4
System reliability	5

quality and quantity of products are guaranteed by this automation. It is possible for people to develop their own personal software in their own way without taking others' use into consideration. However, most software circulates to a certain extent, and also the development and maintenance of it involves many people. This suggests that software is also a kind of product like a material product. Therefore we should assure the quality and quantity of software products by automatic manufacturing (development).

However, software is distinct from a material product because it is a ware that expresses human ideas and has no physical properties. This distinction gives rise to a fundamental way of manufacturing software, and hence an analogy with the manufacture of material products is not appropriate. Although we may learn and acquire useful knowledge for manufacturing software from traditional engineering as used for the manufacture of material products, we need a new engineering based on the concepts and theories which reflect the characteristics of software. For this reason software engineering is expected to be a new discipline. We look forward to developing the framework of software engineering which is different from traditional engineering.

1.2 History and goal of software engineering [4–6]

1.2.1 *Beginning of software engineering*

The need for systematic approaches to the development and maintenance of software has been recognized since the 1960s. During that decade, third-generation computer hardware was developed and influential concepts of software such as multiprogramming and time sharing were proposed. Many software systems were implemented on the basis of these concepts. These capabilities provided the technology for the metamorphosis of batch processing, with the arrival of interactive, online, real-time, multiuser concurrent processing. New applications of computers based on this technology include the command and control of navigational guidance systems, airline reservations, process control and scientific–engineering time sharing. However, some of the original attempts to use these software techniques were never fully realized. Among the systems delivered, there were many examples of cost overruns, inefficiency, lack of reliability, time delay in development and many other problems, and these became a subject of discussion. It had already become apparent that the development of software technology could not keep pace with system development, and this has become a serious problem.

A workshop was held in West Germany in 1968, and another was held in Italy in 1969. The term 'software engineering' was proposed in those workshops. The technical and managerial aspects of the development and maintenance of software became the subject of a discussion with a wide scope.

Since 1968, with the advance of technology and application of computers, research and development in software engineering has been stimulated. The definition and goal of software engineering have gradually become clear and precise.

Software engineering has been defined in many ways. However, these definitions share several common points. The object of software engineering is the development and maintenance of larger software systems than the software produced by one person. The development of those software systems applies several engineering disciplines common to any sort of engineering. Moreover, software engineering deals with not only technical but also managerial issues.

The term 'software' has become common knowledge and is used even in daily conversation. However, this does not necessarily imply that 'software engineering' is used correctly. Software is not simply a computer program related to an application or a product, but it also contains all documents necessary for the installation, operation, development and maintenance of the program. In a large-scale system, to document is as laborious as to develop the program itself.

1.2.2 *Goals and characteristics of software engineering*

The primary goal of software engineering is to improve the quality of software products, including reliability, and to raise the productivity of software engineers. In addition, software engineering deals with social problems originating in software technology. It is based on computer science, communication techniques, systems engineering, management science, linguistics, an engineering approach to problem solving and cognitive science. Concepts and methodologies from these sciences are incorporated into software engineering within the engineering framework, but software engineering is a new technological discipline distinct from the other sciences, and its contents range from the theoretical treatment of ideas and concepts to pragmatic approaches.

There are significant differences between software engineering and traditional engineering. The fundamental causes of these differences are the fact that indispensable physical or chemical laws are unnecessary for software itself and the lack of visibility in the interfaces between software modules. Software has no mass, no volume, no colour, no odour. Software does not degrade with time as hardware does. Only design and implementation errors cause software failures, and degradation does not. Because a program runs on computer hardware, the disciplines used to guide software development and maintenance are constrained by the computer architecture. However, because a program is an expression of human concepts, these disciplines are founded on the principles of intelligent human activity. The principles of intelligent human activity in software are not as strict as Newton's law or Maxwell's equations. Logic is the only strict principle among them. However, logic is insufficient for the above-mentioned disciplines to guide software development and maintenance.

Until now, software engineers have investigated the characteristics of software and have then attempted to formulate principles to guide software development and maintenance. Unlike traditional engineering based on the scientific laws of nature, software engineering looks for principles of software based on laws of intelligent human activity. It also looks for the principles which connect these laws with computer architecture. These principles, if set up, will make automatic software manufacture as realizable as that of material products. This is the goal of software engineering.

1.2.3 *Past results of software engineering [5,6]*

18 years have passed since the beginning of software engineering. All software produced inevitably runs on a von Neumann computer, involving a procedure in which a machine executes primitive operations sequentially. A lot of research and development is applied to this sort of software. A number of notable concepts, which are still useful as the foundation for developing and maintaining current software, were established:

- Software life cycle:
  requirements analysis, design, implementation, testing and maintenance.
- Software quality:
  reliability, understandability, portability, maintainability, testability and usability, and so on.
- Structured programming:
  every procedural program can be written with three constructs – sequence, selection and iteration.
- Software structure:
  hierarchy, modularization, abstraction and information hiding.
- Concurrency:
  prevention of deadlock, mutual exclusion and synchronization.
- Verification:
  logical verification, abstract execution of specifications and prototyping.

- Visualization:
  software representation by diagrams, charts and drawings.
- Software development environment:
  documentation, development assistance, tools for automation and an integrated environment.

With the foregoing fundamental concepts, software engineering has developed the techniques for requirements definition, design automation, testing and quality assurance. Each topic is discussed in detail in other articles and hence this article does not elaborate on them.

In every field of engineering, a product should always be evaluated with quantitative measures. These measures are not always purely technical or scientific. Manufactured engineering products should be evaluated not only with technical or scientific measures but also with measures based on user friendliness or aesthetics. This applies perfectly to software. Well-structured software may be ranked highly with the latter measure as well.

Researches in software engineering have yielded significant results for procedural software. They have improved remarkably the quality and productivity of software. However, they have never caught up with the increased demand for software. An innovational concept or idea is necessary to advance software engineering substantially. Expected directions of the evolution of software engineering under progress are described in the following.

1.3 Vision of forthcoming software engineering [3,4]

Our goal is to improve the quality and productivity of software. Because software manufacture is an inherently labour-intensive activity, this improvement depends on the capability of software engineers and the development management. Both of them have limitations. To improve the quality and productivity beyond human capability, we must realize automatic manufacture of software in the same way as in the production of material goods. If we follow the same track of previous software engineering, it will be difficult to achieve our goal. Therefore we must review, as a start, what is intrinsic in software development.

1.3.1 *Towards a new model of the software life cycle*

In the same way as material products, software has its life cycle, which is the foundation of software engineering. The software life cycle is modelled in several ways. Several new models take into consideration the fact that software demonstrates dynamic evolution as its characteristics are created.

Waterfall model

This is an ideal model proposed in the early days of software engineering, each phase of which requires well-defined input information, goes through well-defined processes, and results in well-defined products. The products become the input information of the succeeding phase. If each phase is completed successfully, software development proceeds without regression. However, this ideal model is difficult to realize in the real world.

Cost model

This modified model reflects the fact that each phase may modify or correct the products of the previous phase, and the cost of each phase takes this modification and correction into account.

Prototyping model

During software development, many decisions are made. In the phase of requirements analysis in particular, as customer's needs are often ambiguous, a simple demonstrative prototype illustrating the functions of the proposed product is helpful to the customer to check whether the needs are fulfilled or not. The manufacture of a material product often utilizes these prototypes. As several techniques for easy prototyping have been developed recently for software, the use of prototypes is becoming popular. Another reason for implementing a prototype is to explore technical issues in developing the proposed product. Sometimes, especially in the human–machine interface, it is impossible to define the product without prototyping.

Successive version model [7]

Product development by the method of successive versions is an extension of prototyping. Many software products including operating systems are developed with this approach. There is also a spiral model essentially similar to this model. In reality, software is evaluated during its operation, and the evaluation is employed later in the development of the successive products step by step. To improve productivity, we must systematize and automate the prototyping and development of single versions of software. We must also re-examine what the life cycle of software is.

1.3.2 *Software development process and its automation [8]*

The software development life cycle consists of phases ranging from requirements analysis to implementation, as described previously. They are partitioned into two processes which are fundamentally different from each other. The first process is to analyse the native problems and to seek the solutions in each application field. The other process is to transform the solution to a form which the computer can interpret and execute. We call the former an upstream process or an analysis process, and the latter a downstream process or a transformation process. Traditional software is procedural; that is, a sequence of instructions is executed by the computer. Both problem refinement and sequentialization proceed simultaneously in software development. Figure 1.1 illustrates this situation. Because the computer executing a program is basically a von Neumann type of machine, a program is a sequential procedure derived from the details of the problem, and it depends on the processing mechanism of the computer. Even if a program is written with a high level language, it is inseparable from the sequential procedure. Thus, traditional software depends strongly on the processing mechanism of the computer. Consequently, both the process to analyse and refine the problem and the process to transform the solution into a program depend on the processing mechanism as illustrated in Figure 1.1.

The analysis process is the first half of software development. In this process, we analyse a problem and seek the solution. Human characteristics such as contemplation or presentation (communication) affect this process significantly. We should think about the issue of whether traditional, procedural programs suit human characteristics or not. The

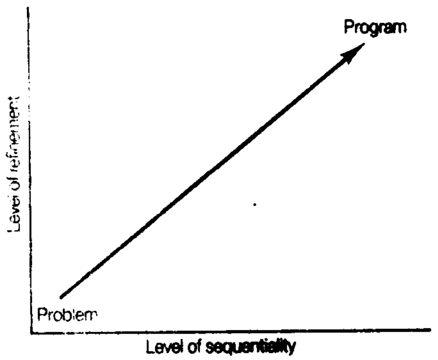

Figure 1.1 *Simultaneous problem refinement and sequentialization.*

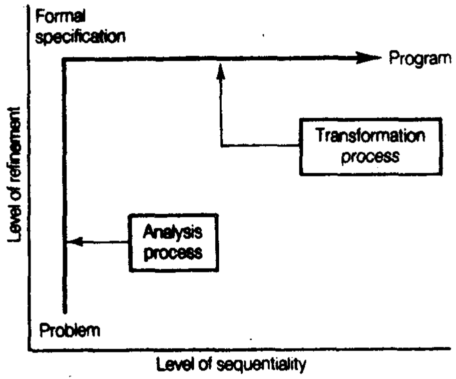

Figure 1.2 *Improved process of software development.*

object of this process is a formal description of the solution to enhance the downstream process, that is, program design and implementation. This formal description should suit human thought characteristics. We call this description a formal specification. A formal specification is a sufficient description which provides enough information for the problem to be solved logically. In principle, it need not have any relationship with the processing mechanism of the computer.

As mentioned above, to solve a problem it is necessary to model it. The human processing mechanism of thought applies this model to analyse a problem. The model is an abstract computation model of human thought. Traditional software development has used a procedural computation model based on computers. If a computation model suitable for human thought exists, there is no need to use a computation model of the machine.

Up to the present, several models for formal specifications have been proposed. Some of them are data flow, finite state machine, functional, algebraic, relational, object oriented, etc. Each model has its own application areas suited to its representation capability.

After the formal specification is formulated, there comes the process of transforming it into a program implementation. This process could be performed in a formalized field of logic. The automation of this process may be realized with tools, if any, generalized or extended from a compiler. If the first process uses a model inexecutable by computer, the latter process must transform it into a computation model. As a summary of the above discussion, Figure 1.2 illustrates the improved processes of software development.

Among software manufacturing processes, the automation of the transformation process is most feasible. In reality, a respectable part of this process has been automated. The initial process of acquiring a formal specification definitely requires intelligent human activity, but a computer can assist this human activity. In the future, most activities of software engineers will converge into this process. Software development will transform into a knowledge-intensive job from a labour-intensive job.

1.3.3 *Computation model and software paradigm (new-type programming)*

The computation model is a theoretical framework, and the computer is supposed to compile, interpret, and execute it. As mentioned previously, most of the traditional high level languages are based on a procedural computation model. Because this model reflects the processing mechanism of computer hardware, it does not satisfactorily reflect a characteristic of human thought. Traditional software engineering has used techniques based on procedural software languages, but these techniques have their limits. To develop high quality software productively, something beyond these techniques is necessary.

Current researches propose new computation models. These models are not limited by any existing computer hardware. They are abstract models of the essence of computation. On the basis of these models, software languages and software development methods now reflect human characteristics more explicitly. New models such as the functional model, logical model and object-oriented model are proposed. The theoretical foundation of these models provides the semantics of software description and the basis of software development methodology. Moreover, integrated models or more advanced models are currently under research.

The methodology for software description and production on the basis of those new models is called the software paradigm. It is also called new-type programming with reference to procedural programming. The software paradigm plays an important role in research on future software. The term 'paradigm' originally meant model, pattern or example. In software engineering, it means models and approaches to solving problems. A software paradigm contains the language of a class (functional, logical or object oriented), a software development environment supporting the language, and software engineering disciplines for creating a system in the environment. Human thought and presentation use different models according to the attributes of the problem. Therefore we must integrate several paradigms into a multiparadigm and seek the most suitable software reflecting human characteristics.

1.3.4 *Knowledge engineering and software development*

Research in artificial intelligence has a longer history than has software engineering. As computer technology advances and fifth-generation computers are investigated, research in artificial intelligence shifts its