Research Trends in Technology for High-Reliability Software — Toward the Establishment of Basic Software Technology —

Masao Watari
Information and Communications Research Unit

5.1 Introduction

Today, 50 years after the emergence of the computer, the device comes in a variety of forms such as supercomputers, personal computers and microprocessors embedded in assorted hardware. As the use of computers has already deeply penetrated into a wide range of activities of businesses, society and individuals, a failure in a computer system has a significant impact. Some of the recent examples still fresh in our memory are the failure in a banking system and the malfunction in a cellular phone system. Banking systems consist of huge and complicated computer systems so as to process via networks various types of transactions requested through automatic teller machines (ATMs). On the other hand, in the case of cellular phones, more complex and larger software programs are used as they not only serve as phones but also offer many other functions including Web browsing, mail handling, special ring tones and photo taking. An increase in the complexity and size of software has led to the challenge of how to ensure reliability.

A system can be reliable only when reliable hardware, reliable software and reliable system administration are combined. Although hardware reliability has been steadily improving with technical advances, there remain a number of problems in the reliability of software, because software production involves a lot of manual work and thus tends to be subjected to instability. In the past age of mainframes (large computer systems), individual software engineers' craftsmanship contributed to the development of large-scale, complex software programs with high quality. Compared to this, today's software has grown even greater in both size and complexity and is more likely to be constructed by combining widely used software components (commercially available or free software) to speed up the development process, specification and interface descriptions between components or between modules are very important for ensuring reliability. Meanwhile, as technologies for software reliability have not made major progress since the mainframe age, it is hoped that new reliability technologies are researched and developed.

This report intends to explain the trends in the research on software reliability technologies and identify the problems to be tackled to improve such technologies in Japan.

5.2 Changes in software production

5.2.1 Software: Moves from "Creation" to "Combination"

In the era of mainframes, software programs were written in different programming languages selected to meet the objective and were run on operating systems that were dependent on hardware systems and specifically designed by individual companies. Under such circumstances there were very few commercially-available software components so that software programs were usually developed from scratch. Later, when hardware became downsized to usher in the age of workstations and personal computers, general-purpose operating systems independent of hardware, such as UNIX and Windows, came into widespread use. More recently, in the Internet era driven by the widespread availability of the Web, applications that are independent of any particular operating system or platform (i.e., applications capable of running on a common virtual machine) have spread.

In the domain of programming languages, where a variety of languages have been used, such as FORTRAN for numeric computations, and COBOL for business transactions, object-oriented languages including C++ and Java have recently become widespread and standard. The spread of object-oriented languages allowed to divide software programs into highly independent software modules so that these made easy to construct software components as well as to distribute and reuse of software. In addition to this, middleware, a type of software that functions as a library shared across applications, has come to be provided for each individual industry field to intermediate between the operating system and applications, so that application programs can simply call and run it. Such progress has allowed software engineers to program by composing "software parts," as if joining blocks. In other words, the software development process has changed from a task of "creation" to that of "combination." The change has allowed even large-scale software to be developed in a short period of time, resulting in an increase in the productivity of software. Figure 1 shows a history of software technology.

5.2.2 Need for improvement of software reliability

(1) Changes in software development

Since many recent software products are required to provide a large number of functions and thereby have become increasingly complicated and enlarged, improvement of their reliability is highly demanded. For example, a cellular phone system consists of over several million lines of program code. Given that one software engineer is said to be able to handle up to about 10,000 lines, development of such software involves a significant number of people. They work at multiple locations of multiple companies, sometimes even including overseas sites to reduce development costs. Thus development process management to maintain good communication among engineers and among different locations is essential, as it determines the quality of the software being developed.

Furthermore, rapidly changing IT product markets call for rapid specification changes, shorter development periods and enhanced system maintainability. Such pressure from the market has in some cases resulted in incomplete testing forced by a short delivery time, careless checking for consistency following a specification change, and insufficient maintenance capability. There is a great risk that development may take

Figure 1: History of software technology

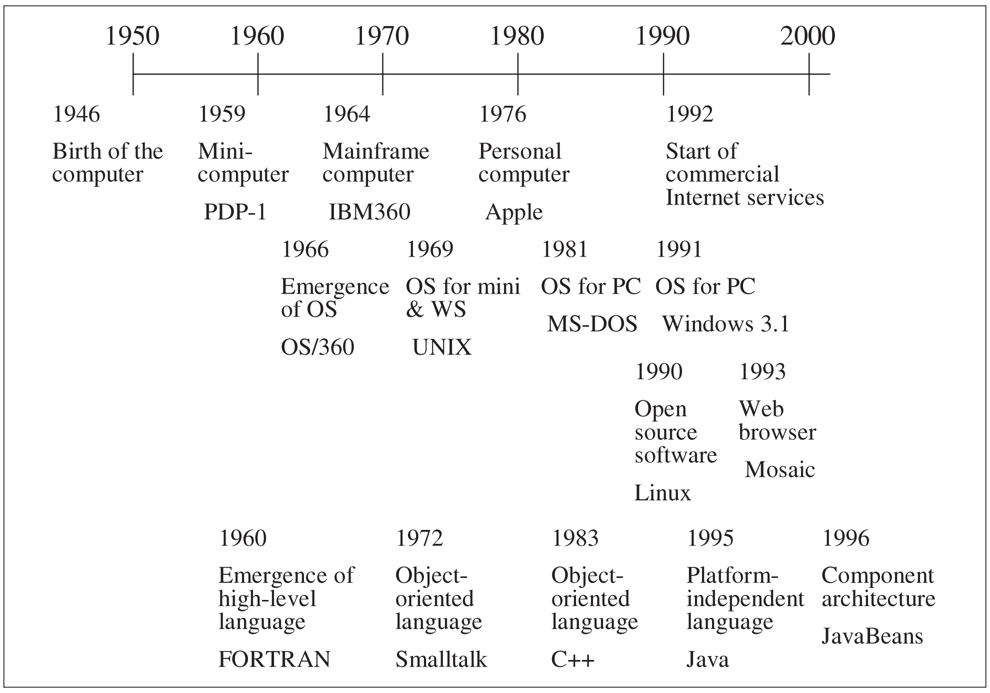
1950
1960
1970
1980
1990
2000
1946 Birth of the computer
1959 Mini-computer PDP-1
1964 Mainframe computer IBM360
1976 Personal computer Apple
1992 Start of commercial Internet services
1966 Emergence of OS OS/360
1969 OS for mini & WS UNIX
1981 OS for PC MS-DOS
1991 OS for PC Windows 3.1
1990 Open source software Linux
1993 Web browser Mosaic
1960 Emergence of high-level language FORTRAN
1972 Object-oriented language Smalltalk
1983 Object-oriented language C++
1995 Platform-independent language Java
1996 Component architecture JavaBeans

Table 1: Factors attributing to lowering software reliability

- Too many functions and too much complexity in a system
- Frequent specification changes
- Short delivery periods that urge careless testing
- Distributed development teams including overseas locations
- Increase in black-box software components
- Increase in software whose quality is not verified (open source software, software components)

Table 2: Software reliability technologies

Design Process	Programming Process	Test Process
UML Formal methods	Object-oriented language	Exhaustive logic test Boundary value analysis State transition test
Process Management		
CMM, TQC		
Development Models/Methods		
Spiral model for software development, XP		

precedence over reliability.

On the other hand, productivity has been increasing through the use of software components that are more widely available in the market. These commercial software parts, however, often cause such problems as an inconsistent interface or incompatibilities with specifications due to version differences. The risk deriving from the use of black-box software components should be well recognized.

(2) Reliability of open source software

Recently, the use of open source software has been growing. It has been attracting attention since Linux, a UNIX-based operating system for PCs, was developed using the open source software method in 1990. Following this, basic software for Internet-based distributed systems and many other software programs have been developed by using the same method. Since as to the open source software its specification and source code are available to the public, anybody can verify its mechanism. Since the open source software is developed by a large number of volunteers who form a community and work in parallel, superior suggestions will meet with quick feedback, leading to a better final product. Since the community members may act as preceding users of the software they have developed, widespread use of the software will be promoted.

Meanwhile, accessibility to software mechanisms has come to be demanded in order to resolve security problems caused by hackers and the like. This is another factor that encourages software product makers to adopt the open source software programs.

While the advantages of the open source software as described above have been highlighted, quality of such software is not ensured because it is basically provided free of charge. There would be no problem in quality if the open source software had undergone extensive testing during the development stage and been tried by many users to improve reliability. However, as open source software is usually developed by a group of volunteers without any compensation, its reliability needs to be verified before use. It should be aware that unverified software will take considerable man-hours for inspection and testing.

The factors that affect lowering software reliability, which were explained above, are summarized in Table 1.

5.3 Software reliability technology

As shown in Table 2, technologies to ensure software's reliability are divided into three fields. Those are support technologies at each process of software development, management technologies for software development process, and models and methods for software development. These technologies are described below.

5.3.1 Object-oriented language

An object-oriented language is a programming language that allows a program and its accompanying data to exist as a self-contained block (called an object). In an object-oriented language, operations are realized through exchanges of messages between objects. In this object-oriented scheme, data can be hidden (encapsulated) within an object so that they can be protected from unwanted access to ensure higher reliability. Reliability can also be improved by allowing clear description of the object's interface, which is written explicitly by the programmer in the object-oriented language.

A programming language, on the other hand, must be flexible because it is expected to be able to describe any kind of specification. The flexibility can sometimes leave room for inconsistency in the interface between objects, resulting in failure to ensure reliability. That is to say, any programming language by itself cannot ensure reliability with flexibility.

5.3.2 System design description

To construct reliable software, a clear design concept and a good grasp of the design among the members involved in the development project are indispensable in any process of software development. As a method to describe system design blueprints, the Unified Modeling Language (UML) was standardized in 1997 by the Object Management Group (OMG), a standardization body for object-oriented languages. Many methodologies for describing systems had coexisted until the developers (See Footnote 1) of the three renowned methodologies developed UML with a view to unifying them into a single standard. UML was comprehensive description by compiling numerous methods that had been used for software design. UML allows the members of development teams to understand the system design in the common language.

UML, as it only provides a method to describe software designs, does not restrict the freedom of design concepts. Therefore, any kind of system can be described, although reliability of a resulting system is not necessarily ensured.

5.3.3 Formal methods

The formal method, a technique that is based on mathematical theories such as algebraic theory, set theory and graph algorithm, can help describe requirements with mathematical precision. In addition, it allows developers to test and verify the behavior of the developed software using mathematical theories. As long as the program's specification is written using a formal method, any specification error detected or any specification change that occurs during development will result in modification that is conducted correctly in mathematical terms. Since specification changes and partial modifications may cause unexpected impacts to other part of the system, such events are considered a major factor of reduced reliability. Under the present circumstances where defects are resolved case by case based on each engineer's experience, how to ensure reliability remains a challenge in using formal methods.

Through research on formal methods, which was initiated in the early 1970s, a variety of methods for description and verification of system specifications have been developed [1],[2].

(1) Formal specification language

A formal specification language, as it allows software developers to write specifications based on mathematical theories, ensures that described specifications are free from errors and conflicts. A number of specification languages of this type have been developed mainly in Europe.

Formal specification languages are divided into two categories. One is the model-oriented language, which is used to describe software specifications by modeling on the basis of set theory and the like, and the other is the property-oriented language, in which data properties are specified based on algebraic theory.

Well-known, model-oriented specification languages include the Z notation (See Footnote 2), developed at Oxford University, Britain, the B-

Footnote 1:
With the intention of standardizing system description methods, UML was developed by Grady Booch, the designer of the Booch method, James Rumbaugh, who advocated the Object Modeling Technique (OMT), and Ivar Jacobson, who proposed Use Case.

Method (See Footnote 3), originated in France, and VDM (Vienna Development Method)(See Footnote 4), designed at IBM's Vienna Laboratory. Among property-oriented specification languages, on the other hand, typical ones are OBJ (See Footnote 5), whose development initiated in around 1977 at the University of California and the Stanford Research Institute (now SRI International), and CafeOBJ (See Footnote 6), an extension of OBJ whose development started in about 1995 at the Japan Advanced Institute of Science and Technology.

Footnote 2:
The Z notation is a language developed at the Programming Research Group at Oxford University in the late 1970s. It is used to specify a system as a state machine (the state space, the initial state, and the operations that affect the state space) on the basis of set theory.

Footnote 3:
The B-Method is a software development tool that has been developed based on the Z notation by Jean-Raymond Abrial, in France, in the mid 1980s.

Footnote 4:
VDM has its roots in the research conducted at IBM's Vienna Laboratory to verify the design correctness of the PL/I compiler in 1974. VDM creates a mathematical model consisting of data sets, lists and mappings, and describes a system specification as changes in the state of the model.

Footnote 5:
OBJ has been developed mainly by Joseph Goguen at the University of California (later at SRI International and then Oxford University) since 1977. OBJ is a formal specification language that expresses abstract data types strictly, using algebraic theory.

Footnote 6:
CafeOBJ, an extension of OBJ, has been developed by an international team led by the Japan Advanced Institute of Science and Technology as a language to specify a system's operations by modeling data and processes uniformly using hidden algebra. It allows to describe or verify specifications of distributed processing systems.

The Z notation, a typical model-oriented language, permits a system to be specified as a state machine (the state space, the initial state, and the operations) so as to let the developer know mathematically in advance all possible states of the system that will take. Using a single space, this method can describe specifications for serial processing systems but not that for distributed cooperative processing systems. On the other hand, CafeOBJ, a property-oriented language, can express data and processes in a uniform manner based on equational logic, enabling to describe or verify specifications of distributed processing systems. A problem in property-oriented languages, which resemble the functional programming language, is their low ability of expression. In summary, model-oriented languages permit expression with relatively high degrees of freedom while restricting analysis and verification. By contrast, property-oriented languages, although lacking expressive power, facilitate analysis and verification.

There are a variety of formal specification languages supporting various mathematical models but they are not unified. Moreover, understanding description written in these languages takes more time because they are written primarily using mathematical notations, which are far away from natural language. For these reasons, formal specification languages have been applied only for the critical areas that require high reliability and have yet to come into wide use.

(2) Checking and verification tools based on formal methods

Researches for solving practical problems by using formal methods are actively conducted in the U.S., and have led to the creation of tools that can check specifications and operations of programs or that can verify such specifications and programs do not cause malfunctions. These tools are divided into two categories. The first category consists of model checking tools based on state searching algorithms, such as SPIN (See Footnote 7), developed at Bell Labs, U.S., in 1980, and SMV (Symbolic Model Verifier) (See Footnote 8), originated at Carnegie Mellon University, U.S., in 1987. The other category is comprised of

Footnote 7:
Originating from the development at Bell Labs, U.S., in 1980, SPIN is a verification tool that performs exhaustive checks over the all possible operations of a system using linear temporal logic, and verifies that the program works properly without any failure. SPIN can be used for the verification of the asynchronous operations in distributed systems.

Footnote 8:
SMV is a checking tool proposed by K. McMillan at Carnegie Mellon University (CMU), U.S., in 1987. It efficiently checks system operations by compressing the subject's state space using the binary decision diagram.

Footnote 9:
PVS is a general-purpose, theorem-proving tool that was developed by SRI International, U.S., in the mid 1980s. Through proofs based on high-order logic, PVS verifies consistency and integrity of specified functions.

verification tools using theorem proving, among which PVS (Prototype Verification System) (See Footnote 9), developed at SRI International, U.S., in the mid 1980s, is well known.

The model checking tools based on state searching, while permitting automatic verification, can only be used for systems that can be described as finite-state transition models. Also, the tools may not be able to run if the target software is too large, because the number of states may increase to a level that causes state explosion. On the other hand, verification tools using theorem proving provide more versatility, although they have not been automated for verification because theorems need to be selected in accordance with the target software's properties.

Similarly, in Japan, researchers at institutions such as the Tokyo Institute of Technology and the University of Tokyo have been working on studies for a method to verify system operations using logical algebra. They developed a method to verify that a system does not stop operating at any type of external input and used it to verify security authentication protocols. There are other ongoing research projects at the Japan Advanced Institute of Science and Technology and other institutions, in which verification techniques for object-oriented model analysis have been developed by extending theorem proving methods, followed by studies on the verification of the state-transition models used for embedded software.

While diverse checking/verification tools have been developed as described above, these tools are not capable of checking/verifying the entire system. An appropriate tool can be selected for checking/verifying a part that should be cut out appropriately from the whole system. The necessity of the basic knowledge about the mechanism of formal methods for using the tools is inhibiting the spread of the tools.

(3) Application examples of formal methods

As shown in Table 3, formal methods often find practical application in the design or the verification of systems in which safety takes precedence over cost. However, it is noticeable that these applications are limited to relatively small-scale systems. What are hoped for toward the future are methods applicable to larger and more complex systems, friendlier to users, and to develop better tools for software environments based on object-oriented languages, which are the mainstream of today.

5.3.4 Testing technologies

Efficient testing techniques are essential because it is said that over half the man-hours for a system development project are necessary for testing. Preliminary arrangements such as embedding checkpoints for testing into programs during the development, are important. As for the test tools, methods have been developed based on mathematics such as algebra and graph theory. Particularly, for object-oriented programs in which operations can be dynamically defined at the time of running the program (dynamic binding), tools based on state transition diagrams have been devised as techniques to test such operations in advance. Some of the testing technologies using graph theory or algebra are described below.

(1) Path test

In this method, control flows are extracted from programs to design test cases. Test cases that cover

Table 3: Examples of formal method application

	Application	Formal Method	Positive Effect
U.S.	Traffic Alert and Collision Avoidance System (TCAS) II	RSML	Formal method used for designing the specifications for collision avoidance
	Fault protection for the Saturn probe	PVS	Verification of the fault protection function
	Program for AT&T's switches	SPIN	Exhaustive testing to identify incorrect operations
Europe	Driverless metro system in Paris	B-Method	Ensured safety
	Air traffic control system in London	VDM	More than a 10-fold improvement in quality
	Customer information management system for IBM Europe	Z notation	Cost reduction by 9% and improved quality
Japan	Railway switch control system	VDM	Ensured safety combined with increased traffic capacity
	Behavior specification for Enterprise Java Beans	SPIN	Identified problems in the description of the EJB 1.1 specification

VDM : Vienna Development Method
RSML : Requirements State Machine Language
PVS : Prototype Verification System

all control flows are generated, so that any part of the source code can be checked at least once.

(2) Exhaustive logic test

Using combinational theory, test cases containing all possible conditions for logic decisions are automatically created.

(3) State transition test

Specifications are described by state transition diagrams and test cases are created for covering all transitions. In the case of a network protocol verification or the like, where the issue of parallel processing is involved, tokens in the Petrinet are assigned values in order to detect parallel processing errors or to analyze the properties, because of avoiding the explosion of the state space.

(4) Data flow test

Control flow graphs are used to detect errors in the data flow.

(5) Boundary value analysis

An analyzer checks whether the system has wrong operations or not at the maximum value, zero and the minimum value.

In carrying out tests sufficient time should be invested for finding and eliminating bugs (program errors) to confirm that the test is sufficient for reducing the bugs to a minimum. Furthermore, in order to test the system efficiently in a limited development period, it is also essential to have a strategy on the test procedure, for example, taking enough time first to test the components critical to the operation of the system, followed by testing the components that are not associated with the system's primary operations.

5.3.5 Software process management techniques

To develop high-quality system software, its development process (software process) needs to be well managed. In the case of the development of a large-scale system, in particular, which is usually conducted by a large number of developers working at multiple distributed places, poor process management will lead to poor-quality software or delays in the development schedule.

Traditionally, in Japan, quality improvements of software are, just as those of hardware, have been driven by the sharing of know-how through small group activities conducted as part of total quality control activities (TQC).

On the other hand, in the U.S., where the mobility of software engineers is high, quality control methods that are strongly dependent on personal capacities have not been successful, producing only a limited effect. Instead of them, the Capability Maturity Model for Software (CMM) has been developed as a technique to accumulate the know-how of quality control with emphasis on organizations rather than individuals. The CMM requires organizations engaged in software development to define the functions required for the level (maturity) of the quality assurance in order to make efforts to improve these levels. By measuring the maturity, the degree of reliability of the organization or the vendor can be evaluated.

The CMM was developed at the Software Engineering Institute of Carnegie Mellon University in response to a request from the U.S. Department of Defense and made available to the public in 1987. Initially, while having found major application in the development of defense-related products, the CMM has become widely used across the world for its ability to provide a measure of software quality improvement. It has already been adopted in over 40 countries including India and China, where a considerable number of organizations are actively implementing the method.

The accumulation of quality improvement know-how promoted through small group activities in Japan, which is a bottom-up approach, has even been reflected into part of the CMM and international quality assurance standards (ISO 9000 series). The CMM requests the improvement of the organization's capacities which should be promoted by top-down activities. It is noteworthy that this turnaround in the approach from bottom-up to top-down was initiated by the U.S.

5.3.6 Software development models

Among other software development models, Extreme Programming (XP) and the spiral model have been receiving attention these days. Although not intended for the improvement of reliability, these new styles of development have contributed to enhanced reliability.

(1) Spiral model for software development

In conventional software development, where the waterfall model was used as the design, coding and testing processes to be carried out step by step, project management was easy. In such an approach, however, it was not until later that errors (bugs) in the specification and the program were found, since the testing process was done after all other processes. In addition, there were often delays in development because programming did not start until the entire specification was determined. Specification changes halfway through development meant rewriting the whole code, resulting in a significant increase in workloads.

In the spiral model (also known as the incremental model), firstly the core of the system is taken out to be designed, coded and tested. Following this stage, peripheral components are developed and added one by one. This allows the core or the most important part of the system to undergo extensive testing from an early phase of the development so that the remaining components can be developed taking the test results into account.

(2) Extreme Programming (XP)

Extreme Programming (XP) has an attention as a new style of development since it was advocated by Kent Beck of the U.S. in 1999 with an eye to quicker development with high-quality. The XP features "paired programming" and "testing first." "Paired programming" is a practice in which a person in charge of program coding from the specifications works together with a person who tests the program. Working this way translates into half the man-hours being spent for testing. "Testing first" indicates that developers design tests that satisfy the specifications before they write the actual code of the program. The coding and testing cycle is repeated until the code passes all the tests. In addition to these two features, the XP has the conventions, such as simple design, collective ownership of the code and open work environments, which have been extracted from past experiences, and requires the members of the development team to practice them to the maximum extent, thus the name "Extreme" came from.

The XP is an excellent test-oriented development model that allows the development of highly reliable software. However, this development model is suited only for relatively small-scale software as it divides a system into units that can be constructed through paired programming.

5.4 Trends in the promotion of research into software reliability technology in Japan, the U.S. and Europe

(1) United States

As the constant leader in software research, the U.S. maintains overwhelming industrial competitiveness. In the area of software reliability technology, DARPA and NSF have been playing major roles in actively subsidizing research projects in military and basic research fields. The technologies accumulated through these research and development activities have been aggressively transferred to industry through collaboration among business, academia and government and have resulted in the emergence of new-generation products. Furthermore, since the 9.11 attacks, investments in security and reliability have swollen due to increased awareness of homeland security.

Specifically, the U.S. government has been actively implementing programs to support research efforts, such as "High-Confidence Software and Systems," a program that intends to ensure reliability, security and safety for mission-critical systems, and "Software Design and Productivity," a program that seeks to improve software cost efficiency through the application of sciences and engineering.

(2) Europe

Although the software industry in Europe is not as brilliant as the U.S. counterpart, the research in Europe is focused on academic methods such as techniques on the basis of fundamental mathematical theories. The research institutions that proposed the Z formal specification language and VDM are both in Europe. Studies on methods to combine the formal description with UML, a language used for specification design, are also actively conducted in Europe.

The EU has also been consistently subsidizing research projects on software reliability technology, as seen in the targets of a range of programs from the European Commission's Information Technologies Research Programme (ESPRIT), which launched in 1985, to the Sixth Framework Programme (FP6), which was started in 2002.

(3) Japan

In Japan, industry has secured reliability technologies by acquiring concepts and research results from the U.S. While many of the past government projects were meant for technical catch-up, recently projects have been launched intending to cultivate original technologies and fundamental technologies. The Japan Science and Technology Corporation (JST), for example, addresses challenges in software reliability based on basic theory through its plans in the category of "Function and Structure" (for the 2000–2005 period) for "Precursory Research for Embryonic Science and Technology (Sakigake 21)," a program intended for nurturing young scientists. The Information-technology Promotion Agency (IPA), in its "Next Generation Software Development" project, which aims to develop, through industry-academia collaboration, software that will become a de facto standard in the global market, supports "High-Reliability and High-Security Software" (2002–2007). Moreover, the Ministry of Education, Culture, Sports, Science and Technology promotes development of technology to produce highly reliable software with high productivity as part of the "Comprehensive Development of the Infrastructure Software for e-Society" (FY 2003-2007) program. The program seeks to construct an IT society where people can participate in with a feeling of security by developing software that serves as the key to building sophisticated information and communications systems.

As described above, Europe focuses on theoretical research while the U.S. stresses research into modeling with a view to application for practical problems. The mobility of researchers between Europe and the U.S. is so high that there has been a trend that theories are studied in Europe and then applied for practical use in the U.S. In Japan, while universities have been taking the initiative in basic researches through small-scale projects to proof their research model, the results have not been applied to practical problems. Their researches have not gone beyond so-called toy models, with little interaction with industry and little effort to address practical challenges. For Japan to join the Europe-U.S.

partnership and expand its contributions to technical advancement, simply presenting theories is not enough, but proposing theories that can provide solutions to practical issues is essential.

Industries are reluctant to conduct research and development of software reliability technology, as it is not likely to bring in profits by itself directly. In Europe and the U.S., the governments have been continuously cultivating this technology as a fundamental technology. Such national-level uninterrupted steady support is needed in Japan as well.

5.5 Challenges to enhance software reliability

What are necessary to enhance software reliability are to develop reliability technology itself and to make full use of high-reliability technology in developing software. In terms of adopting reliability technologies, Japan has already imported the XP, the CMM, both mentioned earlier, and many other technologies from the U.S. When importing technology, however, corporate cultures and professional cultures should be taken into consideration because there are cultural aspects involved in software development and management.

Meanwhile, although reliability technology itself has been making steady progress, further development is hoped for as its current application is limited. Improving reliability in upper processes, where the system is designed, is particularly important, considering that upper process reliability has a significant impact on the succeeding processes. Rigorously checking specification errors during the design process, for instance, would prevent errors from being conveyed to later stages, leading to reduced man-hours for testing. For another approach, if the system specification was described by using a formal method, reliability would be ensured even after specification changes because revisions could be reviewed almost automatically by using formal method verifiers.

In reality, however, formal methods have only been applied to limited cases such as safety-critical systems, as industrial use of formal methods still requires considerable time and effort. There are a number of reasons behind this. The ability of expression of formal specification languages is poorer than that of ordinary programming languages. There are limits in the scale of applicable problems, because a large-scale problem may need a massive volume of calculation. And skills for an formal method are indispensable to describe specifications by the formal method.

Meanwhile, the improvement of reliability is also posing challenges in the domain of “embedded software,” software that is embedded into devices such as home information appliances, automobiles and control systems, which are fields where Japan has a leading edge. Since embedded software has to have real time functions and to take out full power of limited hardware resources, software engineers for its development have been required to have a certain level of craftsmanship. Thus, the meticulous nature of the Japanese has helped in the development of high-performance and high-quality software products. As demonstrated by the fact that of the more than five billion microprocessors (including small processors used for control purposes) produced annually, half use TRON, which originated in Japan, as the operating system, indicating how Japan has been leading the world in the field of embedded software.

Even in the embedded software sphere, however, software programs have come to be required to provide more functions as the performance of hardware enhances, and have been forced to use a growing number of off-the-shelf software components. Since the U.S. has an overwhelming power in the area of software components, the embedded software industry is also now under the influence of the U.S. As the embedded system is usually sold in high volume and thereby a defect would have a significant impact, reliability is a critical element of embedded software products. This suggests that one of the measures Japan could take to maintain its leadership in the field of embedded software is to establish an innovative reliability technology.

A step toward this goal is to promote attempts actively to apply the basic theory of formal methods that has been continuously studied at universities in Japan to practical problems as U.S. has been doing. An effort needs to be made to identify the limitations contained in their fundamental research results and formulate the next innovative theories through the application of such research outcomes to practical problems. The business community in turn would be asked to present real world problems to be jointly studied.

In addition to this, for the purpose of competing with the U.S. research activities in the information technology field, Japan needs to have a greater pool of research talent in the field. According to statistics (See Footnote 10) on the number of academic students in Japan and the U.S., the U.S. surpasses Japan in the number of persons who received in 2000 bachelor's, master's and doctoral degrees in information and communication disciplines (a combined total of Electrical Engineering and Mathematical/Computer Science) by 1.8 times, 3.0 times and 4.7 times, respectively. This suggests the need to increase the number of academic students. On the other hand, China expanding its capacity of graduate schools had 300,000 graduate students (See Footnote 11) for fiscal year 2000, which is well above Japan's same year figure of 210,000. This is a fact that should make Japan realize the need for reinforcement in this area not only in volume but also in quality. In conducting research in information technology fields, Japan has traditionally been playing catch-up with Europe and the U.S., but a time for catch-up is over now that industry is calling for innovative new technologies. Therefore, with respect to the education of scientists, nurturing people who excels at one thing is needed rather than improving average levels or overcoming weaknesses. We should be aware that we are in an age where strong points must be further strengthened and weaknesses should be complemented through partnerships with others.

Footnote 10:

The data for Japan is based on the number of the persons who completed bachelor's, master's and doctoral courses in Electrical Engineering, Department of Engineering, and in Mathematics, Department of Science, in March 2001, and was provided by the Ministry of Education, Culture, Sports, Science and Technology in its survey results on school education (for higher education institutions). Those who left school after finishing doctoral courses without earning any doctoral degree have been excluded. The data for the U.S. is based on the number of the persons who received in 2000 bachelor's, master's and doctoral degrees in Electrical Engineering and Mathematical/Computer Science, and was provided by the NSF Division of Science Resources Statistics.

Footnote 11:

The source is "International Comparison of Educational Indexes" for 2002 and 2003 provided by the Ministry of Education, Culture, Sports, Science and Technology.

5.6 Conclusion

The common feature of products that are highly competitive in the Japanese hardware industry is not low cost but high quality and reliability. Similarly, in the software industry, improvement of reliability is one of the inevitable challenges toward further industrial development. While software programs grow increasingly complex and massive year after year, software products are given shorter life cycles and upgraded more frequently to offer improved specifications. In this context, an environment in which specification changes could be made in shorter periods while ensuring high reliability is demanded. To this end, advancement of reliability technology is hoped to be realized through the use of techniques based on mathematical theories.

In the past, research projects on reliability technology at Japanese universities have tended to end with the establishment of basic theory, without addressing practical problems. Meanwhile, most Japanese companies have imported fundamental technologies from the U.S., so as to develop them into forms that are suited for the Japanese market. If Japan continues to be like this, without having original technologies of its own, the country might lose its competitiveness in the software industry to the U.S. in new technology and to China and India in cost.

What Japan needs to do to have unique reliability technologies is to tackle the challenge of applying the results of its original basic research to practical problems through industry-academia collaboration. When such applied research effort directed toward practical problems requires overwhelmingly greater amounts of research funds compared to the funds for basic research. While industries hesitate about reliability technology research, which would not directly lead to cost reductions, government support becomes vital in this area. Furthermore, if a virtuous cycle could be created in which technical progress from the basics to applications is followed by the feedback to research themes from the application side to the basics, it would truly enhance reliability technology.

Research on reliability technology requires unrelenting improvements supported by sober and continuous effort. Although, in IT-related fields, attention is typically paid to integration technologies to create new business models or new systems, priority should also be given to the endeavor intended for the steady development of basic technology, such as reliability technology, that is underlying IT and exerts an influence on a broad range of fields.

Acknowledgements

The author would like to express sincere thanks to the following individuals for providing their valuable opinions on trends in software reliability technology for the preparation of this paper: Prof. Takuya Katayama and Prof. Kokichi Futatsugi, Japan Advanced Institute of Science and Technology, Prof. Katsuro Inoue, Osaka University, Prof. Kenichi Matsumoto, Nara Institute of Science and Technology, and Prof. Shin Nakajima, Hosei University.

References

[1] E. M. Clarke, J. M. Wing, "Formal Method: State of the Art and Future Directions," ACM Computing Survey, Dec. 1996.

[2] Research Institute of Software Engineering, "Studies on the Strategic Promotion of Software Engineering," Mar. 2002 (in Japanese).

(Original Japanese version: published in March 2003)