情 報 シ ス テ ム 54－7
コンピュータと教育 36－7
（１９９５．５．１６）

CS curriculum in-the-large vs. in-the-small

Toshiro Ohno
Vice President of Technology
Toshiba Information Systems Corp.
7-1 Nissin-cho,Kawasaki-ku, Kawasaki-shi, Kanagawa 210, Japan
Phone:+81-44-200-5141 Fax:+81-44-200-5299
E-mail:ohno@tjsys.co.jp

Abstract

Under the initiative of MITI, a new practitioner oriented Computer Science(CS) curriculum with rather large scopes was established for Japanese software industry in 1994. The scheme was based upon "divide and conquer" strategy targeting upon various problem domains inclusive of both closed systems and complex open systems.

On the other hand, some academic researchers and small software organizations advocated CS curricula with rather small scope targeting upon average programmers paradigm shift based upon recurring concepts like complexity/von Neumann machine/FSM/regular expression etc.

The author compares some experimental results from both approaches and concludes with some guidelines which hopefully assist in average programmers paradigm shift.

1. Introduction

It is estimated that source code scales within U.S. software organizations were 50 times bigger in 1990 than those in 1980 /3/. Basic software like UNIX now needs CD-ROM for installation and Windows NT requires 16MB memory. A typical middleware like DBMS under heterogeneous multiplatform environment has to cope with 3 PC OS, 3 WS OS, 3 DBMS versions, 3 networking software, and 3 or more DFS commercial packages, which results in maintaining 3＊3＊3＊3＊3=243 versions, let alone code patching and customization.

The professional programmers in the 90's have to cope with complexities which have following aspects.

- They have to write far bigger codes in each problem domain.
- They have to implement open systems instead of traditional closed systems by way of multi-layered middleware.
- They have to test each version with combinatorial explosion of test cases.

- There goes a new technology like OOT, and they have to overcome the technological paradigm shift.

From these complexities, it is concluded that professional programmers have to be provided with enough educational opportunities to study CS(Computer Science) which enables their paradigm shift. There seems to be 3 opportunities for Japanese professional programmers to get education/ reeducation.

(1) "Information Technology Curriculum and Courses"(i.e. CS curriculum in-the-large) under the initiative of MITI (Ministry of International Trade and Industry) /1/,/2/.
(2) commercial training courses by vendors like "Microsoft University" and "Novel Advanced Education Center"
(3) OJT(On the Job Training)and off JT within software organizations

(1)and (2)are getting popular these days, but the effectiveness is yet to be seen. Most of (3) are rather traditional, but some new approaches (CS curricula in-the-small) have recently been invented and experimented.

However, the fact is that none of them have gained universal acceptance in Japan . This inertia may mean that organizations are happy to tolerate open system projects failures as an acceptable overhead, or it may mean that organizations find it difficult to assess how a given curriculum can be applied to resolve the problem. This paper describes CS curriculum in-the-small for organizations who wish to understand and resolve their problems in comparison with the CS curriculum in-the-large.

2. CS curriculum in-the-large

In 1994 a new practioner oriented CS curriculum was revised under the intiative of MITI for proffessional programmers in Japanese software organizations. The author participated in the curriculum creation, particularly in the domain of basic software. The courses and examinations based on the currriculum started in autumn 1994. The general approaches to the CS curriculum are shown in Fig.1, and Fig.2.

(1) Project Manager
(2) Systems Analyst(including Auditor)
(3) Application Engineer(for application systems)
(4) Development Engineer(for basic software)
(5) Production Engineer(as superprogrammer)
(6) Technical Specialist(database)
(7) Technical Specialist(network)
(8) Technical Specialist(basic software)
(9) Technical Specialist(software engineering)
(10) System Management Engineer(for facility management)
(11) Educational Engineer(for CS education)
(12) System Administrator(for EUC)

Fig. 1 12 proffessional types

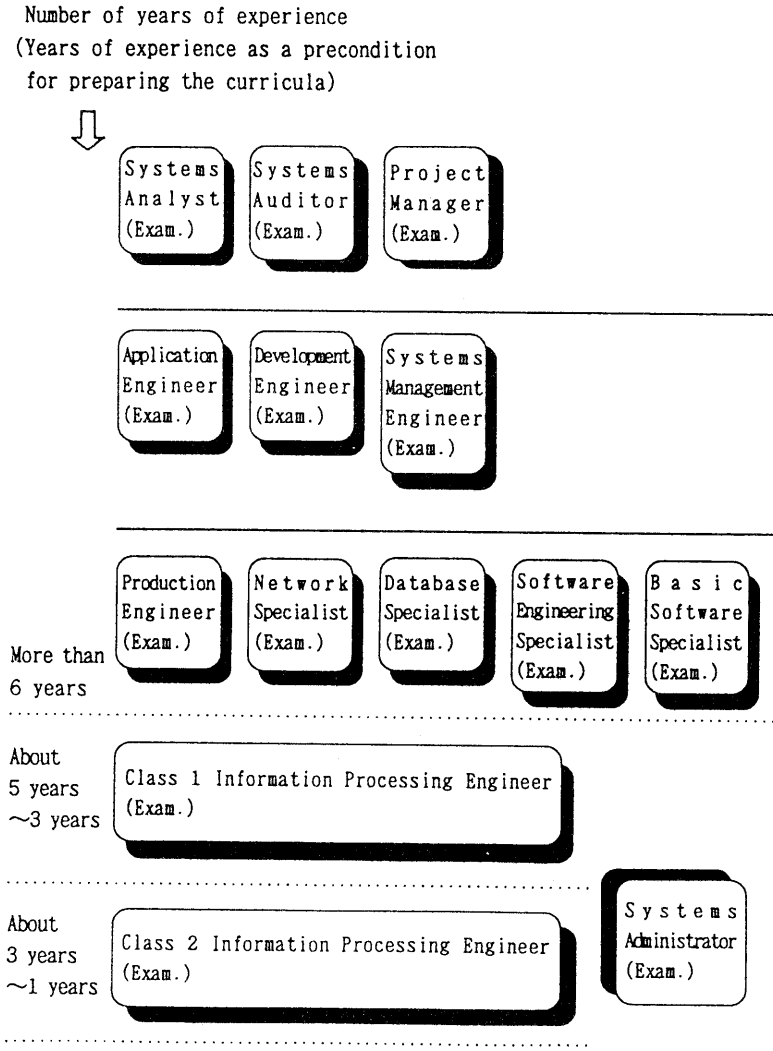

Fig.2 MITI Curriculum Outline

From these it can be seen that the curriculum has several intrinsic assumptions and characteristics with respect to CS education as follows.

(1) Predicted 540 thousand human resource shortage(demand and supply gap) in 2000 should be caught up by educational activities in software organizations based upon the MITI CS curriculum and its courses.
(2) Open systems complexities could be overcome by the "divide and conquer" strategy, i.e. developing a dozen of proffesional types for specific software domains. The breadth of the curriculum is large but the depth is shallow.
(3) The curriculum covers most of the CS domains in ACM CS '88 curriculum, however practical design/ implementation aspect is stressed rather than theoretical/abstraction aspects /4/. The curriculum may well be said example-driven rather than theory-driven. Also, it has little or no explicit consideration for the reccuring concepts as in IEEE/ACM CC '91 /5/.
(4) The curriculum stresses acquisition of technical CS knowledge with little or no references to social processes. There is no substantial OJT in the curriculum. It is lecture-oriented rather than project-oriented. Two of the insufficient social processes are technical writing and touch typing courses.
(5) The curriculum assumes traditional paradigm of software profession, i.e. programming is a problem solving activity rather than a problem finding/formulating activity for customer computing demand.
(6) The curriculum was developed mostly by oldtimers who had lots of imprinting of mainframers closed world. There is little or no references to research and development activities coping with new paradigm of open world. Also, there is no addressing to quality proven free software,freeware, and public domain software.

Over 1500 programmers in the organization where the author affiliated with succeeded in the examinations of the MITI curriculum, however the fact is that they have a hard time to implement client-server systems to customer's satisfaction. Hence, it may well be said that the effectiveness of the curriculum toward open system environment is yet to be seen among software community.

3. CS curriculum in-the-small

Instead of broad but shallow MITI curriculum, several experimental research activities have been conducted by academia and by small software organizations aiming at

proffessional programmer empowerment based upon small but deep curricula.

3.1 Programming curriculum for housewives

The organization of the curriculum is shown in Table 1. The experiment was conducted by Dr.H.Ohiwa(Keio University) for 50 novice housewives who were selected among over 100 housewives through "programmer aptitude test" by A.Munzert /6/. The introductory course stresses computer literacy. The elementary course focuses upon programming. The advanced course emphasizes actual development process through the experimental mini-project. The mini-project consists of the exercise implementing "account book program" focusing upon programming style, design review, stepwise refinement, and module reuse/ extensibility. About 40% of the housewives succeeded in the exercise yieding good codes and could understand fairly well CS concepts like von Neumann architecture, regular expression and finite state machine.

Table 1 Organization of the curriculum

courses	contents	no.lectures	hours
introductory	touch typing editor C	3	15
elementary	control structure array/function programming style user interface	6	30
advanced	pointer structured data file mini-project	6	30

After the evaluation of the result, Dr.H.Ohiwa proposed the following curriculum for novice programmers in small software organizations.

(1) exercise in programming and in programming style (100H)
(2) design and implementation of formula translation (50H)
(3) study of finite state machine/ regular expression (50H)

3.2 Curriculum for students in CS department

The course developed by Dr.I.Kimura(Tokyo Institute of Technology) aiming at educating future software proffessionals particularly stressing personal growth /7/. It has been conducted several years and highly evaluated by academic fields. The outline is as follows.

(1) It is built around a "fuzzy" problem, of which the definition itself is unclear.
(2) An overall understanding by the students of the various aspects of software development is aimed at by exercises simulating real experiences.
(3) Supplementary training is given for writing a good technical paper, keeping journals successfully, Supplementary training is given for writing a good technical and improving typing skills.
(4) The students' stamina is enhanced by creating a "no time limit" environment to go home before the students reach a "perfect" solution to the problem of the day.

The point (4) seems to be particularly important in the present Japanese social situation where some students,by being given too much parental protection, tend to remain childish. This course offers an answer to the tradition in Japan's information systems community to discount the value of CS education in universities.

3.3 OOA/OOD/OOT curriculum for average programmers

The experiments were undertaken by the author in order to compare and evaluate two curricula - the one with recurring concept and the other without /8/. 10 average programmers(A group) took 3 weeks lectures on the follwong programming methodologies chosen as typical recurring concepts.

- CSP(Communicating Sequential Process)
- FSM(Finite State Machine)
- TWS(Translator Writing System)

10 average programmers (B group) did not take any lectures beforehand, however they are well acquainted with traditional structured programming. Both groups took 3 weeks C++ programming course particularly focusing upon the concepts like abstract data type/ inheritance/ polymorphism.

Then, following 3 problems were given as OOT(Object Oriented Technology) excercises for both groups.

(1) library system
(2) lift control system
(3) text processing system

The result is shown in Table 2. Most of programmers in B group had a hard time to understand and to implement right classes and right class library structure. They were good at implementing "has-a" structure (component), but poor at implementing "is-a" (class) structure. They could not devise appropriate CSP/FSM/TWS algorithms and data structures by themselves. Some programmers in A group also could not understand these concepts in spite of given lectures. In the case of library and lift problems, most programmers in B group found it difficult to distinguish real world modelling from implementation, which is a salient feature of JSD(Jackson Systems Development) based upon CSP.

Table 2 demonstrates the importance of recurring concepts i.e. essential methodological principles underlyng in each specific problem domain. These principles seem to have little or no close relationship to acquire general OOT concepts like abstract data type/inheritance/polymorphism.

Table 2 Success rate of 3 problems

problems	success rate(%)	
	A group	B group
library	80	30
lift	70	50
text processing	90	50

3.4 Lessons learned

The above 3 approaches were developed for use by any size and type of software organizations at all stages of maturity. Therefore, the curricula are rather small but the course contents are deeper in each problem domain. The approaches assume that programmers should have some basic underlyng competence before getting into traditional teaching-learning type curricula inclusive of MITI courses and vender provided courses.

The competence covers following subjects which can be taken for a general guideline as the petty curriculum for average programmers.

(1) computer literacy education inclusive of typing andtechnical writing
(2) actual programming experience
(3) understanding of recurring concepts like CSP/FSM/TWS

The approaches also stress project experiences rather than CS knowledge aquisition activities. The mini-project experiences imply both people issues and social process issues distinct from traditional teaching-learning type curricula. The education-study type curricula and courses seem to lead to more competent proffessional programmers.

4. Conclusion

The focal point of the broad but shallow curriculum by MITI initiative is in the "divide and conquer" strategy for controlling exploding complexities in open systems environments. The relevant educational activities are now proliferating among software organizations in Japan, however their effectiveness seems to be yet to be seen.

The good points of the narrow but deep curricula by academia initiative lie in the following. (1)It is easily assimilated by almost all software organizations. It is not threateningly complex and can adapt to any size and tye of software organization. (2)The effectiveness of the guideline was already demonstrated in some specific problem domains yielding competent proffessional programmers.

The poor point of this petty curriculum lies in the less popularity in Japanese software community. Oldtimers, especially managers, in these organizations hopefully can afford to gain necessary paradigm shift through the petty curriculum, or we should await alteration of the generation.

References

(1) MITI/CAIT: New information technology engineers curriculum, CAIT,Tokyo 1995

(2) MITI/CAIT: New information technology engineers texts,CAIT,Tokyo 1995

(3) Information Week: "What's in store" May 23,1994

(4) P.Denning: Computing as a Discipline, CACM vol. 32 no.1,1989

(5) A.B.Tucker(ed.): Computing Curricula 1991, ACM/IEEE-CS NewYork,1991

(6) H.Ohiwa: A proposal for fundamental software education in enterprises, Engineers no.553, JUSE,1994

(7) I.Kimura: A laboratory course for future software proffessionals stressing personal growth, Journal of Information Prossessing, IPSJ, 1993

(8) T.Ohno: OOA/OOD/OOP case studies , bit vol.22 no.11, 1991