International Journal of Engineering Education Vol. 27, No. 1, pp. 70–76, 2011
Printed in Great Britain

0949-149X/91 $3.00+0.00

Research and Practice on Education of SQA at Source Code Level*

YAN-QING WANG,[1] ZHONG-YING QI,[1] LI-JIE ZHANG[2] and MIN-JING SONG[1]
[1] School of Management, Harbin Institute of Technology, Harbin 150001, P. R. China. E-mail: yanqing@hit.edu.cn, qizy@hit.edu.cn
[2] School of Software, Harbin Institute of Technology, Harbin 150001, P. R. China

The exponential increasing in software size and complexity is miring testers with endless testing work and assuring software quality becomes more difficult. A pedagogical model was proposed for SQA at source code level, in which coding standards and code optimization were determined as two quality buses while code review traversed almost the entire software development process. In order to consolidate this model, some topics were researched such as evaluation index system of coding standards, evaluating website of coding standards, process improvement and behavior analysis of peer code review, and so on. The model was developed over four academic years of practice; it is possible for students to eliminate defects in their programs at source code level.

Keywords: software engineering education; software quality assurance (SQA); source code level; coding standards; peer code review (PCR)

1. Introduction

The internet and information technology revolution are accelerating the increase in software size and complexity making the software crisis problem, yet to be well solved, more rigorous [1]. As the major approach to solve the crisis problem, software quality assurance (SQA) comes to be popular. However, SQA has not played its role as we expected. Three reasons can be addressed as follows: (1) software quality is largely dependent upon software designers, developer and testers. The attitude and habits of project members are the most critical factors; (2) although testing techniques are evolving, software size and complexity are increasing even more quickly; (3) testing is too expensive for many software companies to afford.

Therefore, finding an efficient and economical SQA approach is a high priority for both the global software industry and academia. SQA research can be classified as *testing-based SQA* and *process-oriented SQA*. SQA at source code level (SQA @Source) is a concrete implementation of *process-oriented SQA*.

Testing techniques have been developed greatly in recent decades.

1. *V-model* and *test-driven development* [2] have been applied widely;
2. Testing criteria are fully developed and used to identify the testing use-case systematically [3];
3. *Objective oriented testing* and *component-based testing* are well developed and improved [4];
4. *Protocol testing* and *reliability testing* as the supplementary ways have achieved a lot [5–6].

However, testing alone cannot solve the software quality problems so that researchers proposed methods emphasizing quality assurance during development process instead of during testing. Everett proposed *developing axiom* that pointed out that software quality was not determined by testing but system analysis and development [7]. Basili and Boehm claimed that most of defects were injected in design and coding phases [8]. To remove defects, testing cost was much higher than that of code review. Moreover, Prof. Humphrey at CMU proposed zero-defect objective. He proposed that *the more defects are found when testing, the more defects will be missed after testing* [9].

Engineering education is the prophase of industry. Most of students in universities will be professionals in industry. Qualified software programmers require specific training in SQA [10] so our research was focused on the education of *SQA@Source*.

The paper is organized as follows. Section 2 presents related work by others. A pedagogical model of *SQA@Source* is introduced in Section 3. Since we made much more progress on *coding standards* and *peer code review* (PCR) than other topics, our methods and cases on these two topics are summarized in Section 4 and 5. Finally in Section 6, concluding marks are made and future work is introduced.

2. Related work

Process-oriented SQA has become very popular. The relevant techniques *coding standards*, *code review* and *static analysis* are utilized to assure software quality at source code level.

* Accepted 2 September 2010.

2.1 Coding standards

Complying with coding standards has been debated many times in utilizing programming languages, especially under pedagogical background. In 2001, Fang prompted the existing problems in source code of software products, and addressed the problems by coding-standards-based quality assurance model [11]. Dr. Li proposed an effective approach of teaching coding standards in programming [12]. Kirsti et al developed an evaluating tool for the beginners and created a pedagogical model on coding standards [13]. Moreover, some famous software companies such as *IBM* and *Microsoft* built up coding standards for some specific languages; *Parasoft* and *TIOBE* developed code checkers to assess coding standards.

However, many problems still exist as follows:

1. Theoretical system on coding standards has not been built up;
2. There are few effective approaches to measure how much students or professional engineers comply with coding standards in their programming practice;
3. There is no available evaluating website for students or engineers to evaluate their own programs.

2.2 Code review

Code review has been attracting lots of research. Early in 1996, Belli and Crisan presented the automation of code review [14]. Later on, Jun-Suk Oh et al demonstrated the automated code review prior to manual code review [15]. In 2006, Dr. Li presented an approach of using peer code review to evaluate coding standards [12].

Code review was utilized as an effective approach in SQA. Nevertheless, there is not a well-defined model especially for *PCR* and the process has not been researched systematically.

2.3 Static analysis

Static code analysis is the analysis of computer software which is performed without actually executing programs built from that software. In 2006, Geay et al developed a static analysis tool targeting lightweight program verification and finding coding defects in Java [16]. Seth et al described an available tool of static analysis [17]. Some static analysis tools for Java are very popular such as *FindBugs*, *PMD* and *Checkstyle*.

Although many academics and engineers have done lots of research on *SQA@source*, the theoretical system has not formed yet. The related research has not been concentrated so that their reference value has not been brought out as expected.

3. Pedagogical model of SQA@source

3.1 Model definition

After several years' work on *coding standards* and *PCR*, the pedagogical model of *SQA@Source* has been proposed. It is based on traditional software lifecycle [10] with prior stage *design* and following stage *unit testing*. It relies on two buses (*coding standards* and *code optimization*) and one thread (*code review*) (See Fig. 1).

3.2 Model description

3.2.1 Coding standards and code optimization

Coding standards and *code optimization* play their roles in the whole model. The other stages are all performed under the constraints of *coding standards* and the guidance of *code optimization*.

3.2.2 Code review

It is divided into three stages in our model: *self code review*, *peer code review* and *tutor code review*.

1. *Self code review* is a personal oriented process. Being assisted with *code review checklist* [18], a student may review and revise program to comply with *coding standards* and to conform to *code optimization* principles. *Self code review* should be done before the first compile [18].
2. *Peer code review* involves two players. In this process, one student is author and another is reviewer. Every participant will be both author and reviewer. Besides commenting for the remained defects in programs, *coding standards* and *code optimization* are still the main concerns by reviewers.
3. *Tutor code review* is the last stage of code review, in which tutor (instructor or supervisor) reviews the code written by students. Some missed defects may be addressed and suggestions on *coding standards* and *code optimization* may be presented.

3.3 Static analysis

Static analysis will be undertaken after *coder review* and before *unit testing*. Some successful methods such as *input processing*, *buffer overflow prevention*

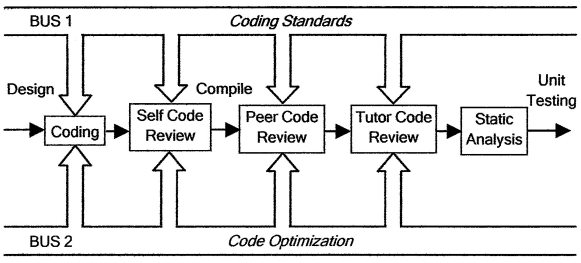

Fig. 1. Pedagogical model of SQA@Source.

and *exception management* will be utilized in this stage. Static analysis has become a very helpful technique to make secure and reliable software products.

4. Coding standards: methods and cases

Along this topic, our work covered *simplified coding standards*, *an evaluation index system*, *a case study*, *a web-based evaluating platform* and *a student-centered educational model*.

4.1 Quality outlook of coding standards

To emphasize the importance of *coding standards* and measure the compliance degree of our students, the research on *coding standards* was commenced with the students who majored in software engineering. The related paper, *Quantitative Research on How Much Students Comply with Coding Standard in Their Programming Practices*, was published in Proceedings of the 3rd China Europe International Symposium on Software Industry Oriented Education held at Dublin on Feb 6–7, 2007.

Also, in order to minimize the conceptual confusion, the difference between *programming style* and *coding standards* was discussed and the usage of *coding standards* was strongly recommended [19].

4.2 Simplified coding standards

Many famous software companies, such as IBM and Microsoft, have their own sets of *coding standards*. Since hundreds of rules and guidelines in one set of coding standards are difficult to follow by students who are studying their introductory programming languages, a simplified version of coding standards (including four sections *naming*, *layout*, *comment* and *coding*) was presented for college students.

4.3 An AHP-based evaluation index system

Based on the simplified coding standards, the hierarchy of evaluation index system was constructed, which included 3 layers (*target*, *criterion* and *index*) and nineteen indices (See Fig. 2 and [20]).

There is no doubt that different index has different weight. A questionnaire approach was applied to collect information about indices' weight. Then analytical process hierarchy (AHP) was utilized and the weight distribution was acquired [20] (See Table 1).

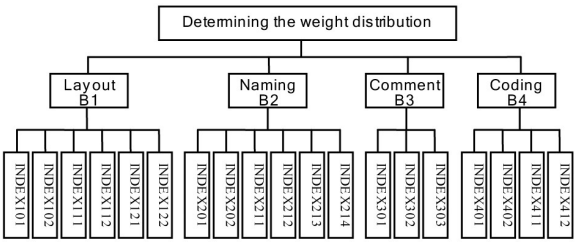

Fig. 2. Structure of evaluation index system.

Table 1. Weight distribution of indices

Layout	**101**	**102**	**111**	**112**	**121**	**122**
	2.58	3.32	4.19	3.74	6.02	3.29
Naming	**201**	**202**	**211**	**212**	**213**	**214**
	4.60	4.22	6.23	2.92	6.07	2.34
Comment	**301**	**302**	**303**			
	6.88	8.44	11.86			
Coding	**401**	**402**	**411**	**412**		
	6.45	4.23	3.65	8.97		

4.4 A case study on coding standards

With the evaluation index system, a case study was done to make sense how many students were ready to write programs complying with coding standards.

4.4.1 Data collection

In 2007, all 159 frosh in school of software at Harbin Institute of Technology were involved in our research. The experiment was taken in the course of *Personal Software Process*. In this course, students were required to submit 5 programs about 100 lines of code each at a certain interval. Theoretically, 795 programs would be received. Since several students submitted less than five programs, 675 samples were collected eventually.

4.4.2 Experiment

The 675 programs were input into a Java application. Based on the evaluation index system, the mark of each program was computed and stored into a MySQL database.

4.4.3 Results analysis

With the data in database, the mark distribution was calculated and summarized. From the mark distribution, it was found that the evaluating result of each assignment was a normal distribution. The average marks of 5 assignments were calculated (See Fig. 3).

Obviously, the data in Fig. 3 were not so satisfactory. The majority of students got marks in the range from 60 to 70. The average marks of the five assignments had no significant difference from 1 to 5 sequentially. After analyzing and summarizing, three major reasons were addressed as follows:

1. Few instruction on *coding standards* in other programming courses;
2. No timely feedback on *coding standards*;
3. Lack of consistent training on *coding standards*.

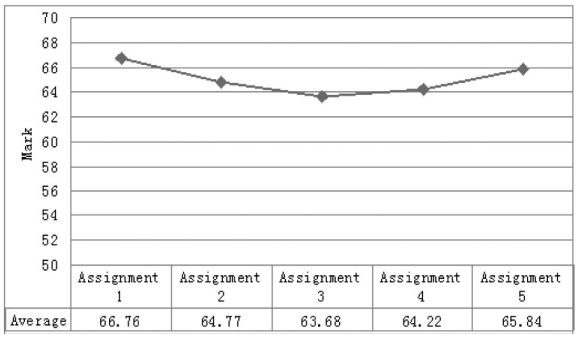

Fig. 3. Average marks of five assignments.

In order to solve the above problems, a web-based management information system (MIS) was developed and a coding standards oriented model was built up.

4.5 A web-based evaluating platform

So as to facilitate students to evaluate their work consistently and get timely feedback online, an evaluation website was constructed, with which students could upload their programs, get benchmarking results, and find detailed shortcomings on *coding standards* [20].

4.6 An educational model

With the AHP-based evaluation index system, the web-based evaluating platform, and the above case study, an educational model was conceived to enlarge the learning outcome on coding standards. The related paper, *Teaching Model of Coding Standards Based on Evaluation Index System and Evaluating Platform*, was published in the Proceedings of 2008 International Conference on Computer Science and Software Engineering held at Wuhan on Dec 12–14, 2008. In this model, *teachers*, *students*, *software industry* and *evaluation platform* were involved. It was a student-centered model, which required all sides interact with each other positively (See Fig. 4).

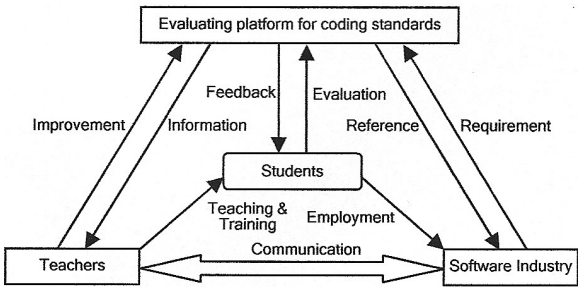

Fig. 4. Coding standards oriented educational model.

5. Peer code review: methods and cases

Along the topic *PCR*, our research included *PCR process refinement*, *behavior analysis of practitioners based on a case study, grouping strategy, game theory modeling* and *a web-based MIS*.

5.1 Kickoff research on PCR

A preliminary PCR model was put forward in 2007. After the data was collected from a 14-student class in C++ course, some problems were addressed. The paper, *How to Evaluate Students' Learning Outcome: A Peer Code Review Model in Undergraduate Computer Programming Class*, was published in the Proceedings of the 2nd International Conference on Computer Science and Education held at Wuhan in July, 2007.

5.2 Process refinement

In order to make the process more clear and facilitate future research, the previous PCR process was refined [21]. In the new model, four roles (*author*, *reviewer*, *reviser* and *instructor*), three documents (*manuscript code*, *comments form* and *revision code*), and the process were all redefined (See Fig. 5).

Roles were redefined as follows:

1. *Author* is a student who writes program and waits for review activity by someone else;
2. *Reviewer* is a performer who reviews the code written by an author;
3. *Reviser* is the author who does revision after receiving comments from reviewer;
4. *Instructor* is the lecturer or teaching assistant who takes responsibility for the quality of laboratory.

Documents were reclassified as *manuscript code* (first edition of source code by a student), *comments form* (written comments by reviewer), and *revision code* (final edition of source code by a student).

Process was clarified as in Fig. 5 which covered six

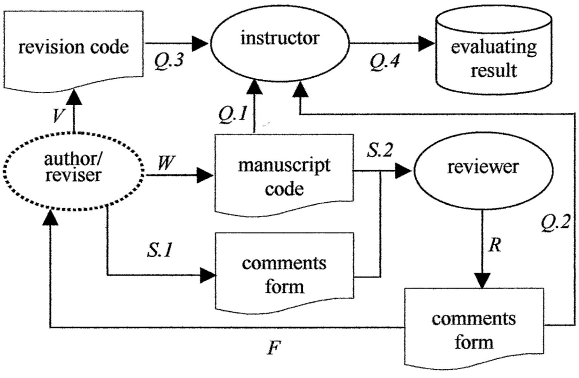

Fig. 5. Flowchart of the improved PCR process.

phases in order: *write* (letters *W* and *Q.1*), *submit* (letters *S.1* and *S.2*), *review* (letter *R*), *feedback* (letters *F* and *Q.2*), *revise* (letter *V*) and *quality assurance* (letters *Q.1*, *Q.2*, *Q.3*).

5.3 Case study on PCR

Afterwards, a case study was completed to check the students' performance in PCR process [21].

5.3.1 Experiment

The experiment was taken at the *Object Oriented Programming* laboratory of HIT-DIT class in spring semester of 2007. This class was from a joint programme developed by School of Software at Harbin Institute of Technology and the School of Computing at Dublin Institute of Technology starting from September 2003 and its objective was to cultivate more qualified industry oriented undergraduates. All nine students were required to submit their programs according to process in Fig. 5 through email. Finally, 178 emails were received.

From the results, it was found that the design of PCR process had shortcomings and email was found not a manageable submission way. *Careless authors*, *irresponsible reviewers* or *busy instructors* might discount the learning outcome of the entire PCR process and push it in a bad way.

Later on, interviews with four students were made separately and some conspiracy issues were discovered.

1. Author did code review instead of reviewer;
2. Students performed a private review in advance of the formal PCR process;
3. Author copied code from a good programmer and did a minor modification, etc.

5.3.2 Behavior analysis

After the case study, the behavior of all participants was analyzed to address the crux [22].

1. ***Author***. Some improper behaviors by authors were found. For example, some author students did not write their code carefully but sent to reviewer even without compiling or self review. Also, after receiving comments from reviewer, some authors scan it briefly and made only a few trifling revisions.
2. ***Reviewer***. The behavior of reviewer was diverse too. All positive and responsible reviewers could finish review work consistently while other reviewers might not always devote themselves into their roles. Moreover, when reviewing poorly written program by a low-level student, the reviewer often finished it unwillingly.
3. ***Instructor***. Inspection was a time consuming work. It was nearly impossible for instructor to inspect all submissions with every assignment by every student. Also, some reviewers often confused instructor with poorly written comments.

5.3.3 Findings

From the result of behavior analysis, it was found that the stand or fall of this game depended on many personality factors. To achieve a better learning outcome, a control mechanism had to be built up to normalize the behavior of every game player. Therefore, grouping strategy and game theory model were introduced in our later research.

5.4 Grouping strategy of PCR

According to the problem findings and behavior analysis in case study, it was concluded that grouping strategy was one major crux of the problems in PCR.

After implementing *pair review* (2 students a group) in 2006 and *circle review* (3 students a group) in 2007, some new grouping strategies were discussed as follows:

1. Switching a *2-student* or *3-student* grouping strategy to *n-student* grouping strategy, in which the *ring-wise review* approach could be adopted. However, the complexity would increase and the effectiveness might reduce as usual after a period of time.
2. Switching a *fixed* grouping strategy to a *random* grouping strategy. This would also increase the management cost because instructor has to generate grouping result for each assignment.
3. Switching a *P2P* (one reviewer to one author) strategy to a *T2P* (multiple reviewers to one author) strategy. But it might be not practical because of its high expense.
4. Introducing *double blind review* mechanism. It could solve the conspiracy problems better because any student could not know who his/her partner is.
5. Exploring a *ranking policy* when grouping. Students would be grouped by the ranking of their programming capability. If the capability difference between author and reviewer is too big, the performance would go down definitely.

5.5 Game theory modeling

In two academic years of implementation, it was found that the PCR process was a 3-party repeatable game and ethical issues were the key that determined its success or failure. Based on two premises, a game theory model was constructed and the suggestions of applying the model were presented as well [22].

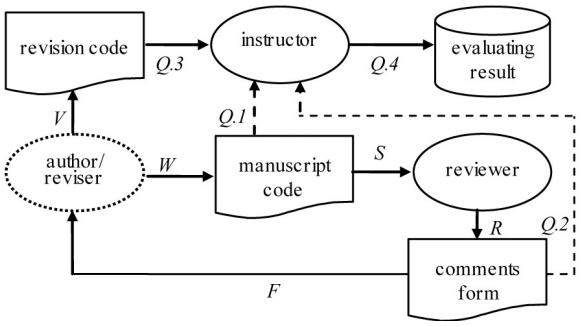

Fig. 6. The web-based PCR process.

5.6 A web-based MIS

In order to make our research and pedagogy more practical, a website built in blind review mechanism was constructed [23]. Students and instructors could do their work on this platform conveniently while the quality could be assured in great measure. Being supported by website, the process was further simplified and the activities *Q.1* and *Q.2* were done automatically by system (See Figs. 5 and 6).

6. Conclusions and future work

Although the development of software testing takes on a pleasing look, the increasing speed of software complexity and size is leaving that of testing techniques far behind. Unfortunately, the trend is going on. That is why the education of *SQA@Source* has been drawing our pressing concerns.

A pedagogical model of *SQA@Source* was put forward. In this model, such main topics as *coding standards* and *PCR* were researched. Based on the work, the case studies were undertaken and the analysis results uncovered more facts for us. However, the model has not been completely explored by us so far. For instance, we have just some preliminary ideas about code optimization. Anyway, the research and practice summarized above might show their reference value in software education area and are leading us towards more interesting topics.

Actually, there is lots of work to do along the research on education of *SQA@Source*. For examples on coding standards: (1) measuring quality of identifier naming is a very meaningful topic; (2) measuring of comment quality is another challenging question; (3) a more feasible evaluation index system should be reconstructed, which should be compatible with the sets of coding standards by different organizations such as *GNU*, *Java*, *ANSI*, *Linux* and *K&R*. And about PCR, the game theory model and practical grouping strategies should be melt into the web-based MIS to make the process more feasible.

References

1. S. Murugesan and Y. Deshpande, *Web Engineering: A new Discipline for Development of web-based systems*, Springer-Verlag, 2001.
2. D. Janzen et al, Test-driven development concepts, taxonomy, and future direction. *Computer*, **38**(9), 2005, pp. 43–50.
3. H. Zhu. A formal analysis of the subsume relation between software test adequacy criteria, *IEEE Trans. Softw. Eng.*, **22**(4), 1996, pp. 248–255.
4. R. V. Binder, *Testing Object-Oriented Systems Models, Patterns, and Tools*, Addison Wesley Longman, Reading, MA, 2000.
5. G. V. Bochmann and A. Petrenko, Protocol testing: review of methods and relevance for software testing, *Proc. ACM/ SIGSOFT Int. Symp. Softw. Testing and Analysis*, 1994, pp. 109–124.
6. J. Poore et al, Planning and certifying software system reliability, *IEEE Software*, **10**(1), 1993, pp. 87–99.
7. G. D. Everett and R. McLeod, *Software Testing: Testing Across the Entire Software Development Life Cycle*, IEEE Computer Society Press, 2007.
8. B. Boehm and V. R. Basili. Software Defect Reduction Top 10 List, *IEEE Computer Society*, **34**(1), 2001, pp. 135–137.
9. W. S. Humphrey, *PSP: A Self-Improvement Process for Software Engineers*, Pearson Education Asia Ltd., 2006.
10. R. S. Pressman, *Software Engineering: A Practitioner's Approach, (6e)*, McGraw-Hill, 2005.
11. X. F. Fang, Using a coding standard to improve program quality, *Proc. of the 2nd Asia-Pacific Conf. on Quality Softw.*, IEEE Computer Society, Los Alamitos, CA, USA, 2001, pp. 73–78.
12. X. S. Li, Using Peer Review to Assess Coding Standards—A Case Study, *Proc. of 36th ASEE/IEEE Frontiers in Educ. Conf.*, San Diego, CA, 2006, pp. 9–14.
13. A. Kirsti et al, Supporting Students in C++ Programming Courses with Automatic Program Style Assessment. *Journal of Information Technology Education*, **3**, 2004, pp. 245–262.
14. F. Belli and R. Crisan, Towards automation of checklist-based code-reviews, *Proc. of 7th Int. Symp. on Softw. Reliability Eng.*, IEEE Computer Society, White Plains NY, Oct 30— Nov 2, 1996, pp. 24–33.
15. J. S. Oh and H. J. Choi, A Reflective Practice of Automated and Manual Code Reviews for a Studio Project, *Proc. of the 4th Annual ACIS Int. Conf. on Computer and Information Science*, IEEE Computer Society, Washington DC, USA, 2005, pp. 37–42.
16. E. Geay et al, Continuous Code-Quality Assurance with SAFE, *Proc. of PEPM'06*, ACM, Charleston, South Carolina, USA, January 9–10, 2006, pp. 145–149.
17. H. Seth et al, Uprooting Software Defects at the Source, *ACM Press QUEUE*, **1**(8), 2003, pp. 64–71.
18. W. S. Humphrey, *Introduction to the Personal Software Process*, Addison-Wesley, Reading, Mass, 1997.
19. Y. Q. Wang et al, Complying with Coding Standards or Retaining Programming Style: A Quality Outlook at Source Code Level, *Journal of Software Engineering and Application*, **1**(1), 2008, pp. 88–91.
20. Y. Q. Wang et al, An AHP-based Evaluation Index System of Coding Standards, *Proc. of 2008 Int. Conf. on Information Tech. in Educ.*, Wuhan, China, December 12–14, 2008, IEEE Computer Society, pp. 620–623.
21. Y. Q. Wang et al, Process Improvement of Peer Code Review and Behavior Analysis of its Participants, *ACM SIGCSE Bulletin*, 40(1), 2008, pp. 107–111.
22. Y. Q. Wang et al, Game Theory Modeling in Peer Code Review Process, *Proc. of 2008 Int. Colloquium on Artificial Intelligence in Educ.*, Wuhan, China, Oct. 17–18, 2008, World Academic Press, England, pp. 113–117.
23. Y. Q. Wang et al, Quality Assurance of Peer Code Review Process: A Computer Science Based Strategy, *Journal of Acta Scientiarum Natralium Universitatis Sunyatseni*, **46**, SUPPL, 2007, pp. 116–120.

Yan-Qing Wang is an associate professor working with School of Management at Harbin Institute of Technology (HIT). He received a B.A., a M.A. in computer science and a Ph.D. in management science and engineering from HIT. He has been a lecturer for seven years in School of Software at HIT. In 2003, he worked with School of Computing at Dublin Institute of Technology as a visiting scholar. His research interests include engineering education, SQA@Source and learning information system (LIS).

Zhong-Ying Qi is a professor, a doctoral supervisor, and a vice dean working with School of Management at HIT. He received a M.A. in management at HIT. Now he is a member of Business Administration Pedagogy Committee at MOE of China. His research interests include *R&D management*, *risk analysis and management*, *industrial strategy management*, etc.

Li-Jie Zhang is an associate professor working with School of Computer Science and Technology at HIT. She received a B.A. in computer application in 1993 and a M.A. in computer architecture in 1999 from HIT. Now she is working as the director of experiment center in School of Software at HIT. Her research covers *software engineering* and *computer application*.

Min-jing Song is a Year 3 student studying in School of Management at HIT. She majors in *Information Management & System* and will continue her study for master degree in two years.