페어 프로그래밍이 직무 성과에 미치는 영향에 관한 연구 : SQL 질의 프로그래밍 성과를 중심으로

윤 성 노* · 김 종 헌* · 박 상 현**

An Experimental Study on Effects of Pair Programming on Task Performance : Focus on SQL Query Programming Performance

Seong No Yoon* · Jongheon Kim* · Sang-hyun Park**

Abstract

In recent years, pair programming has become a widely used approach for development of information systems. According to a worldwide survey, 35 percent of 104 development projects reported using pair programming. However, previous studies have shown rather mixed results in terms of the effectiveness of pair programming, comparing to individual or independent programming. This paper, therefore, uses a lab setting to control some of the variables that appear to have caused conflicting results in earlier studies. Writing SQL queries for given problem statements is selected as the task the subjects to solve. One key issue addressed is the distribution of work load among the pair programmers and the independent programmers. Another is communication among co-workers as would occur in a real-world system development environment. The results of this study indicate there is no significant difference in task performance pair programming and independent programming.

Keywords : Pair Programming, Extreme Programming, Concurrency Metrics, User Perceptions of Pair Programming

논문접수일 : 2007년 08월 24일 논문게재확정일 : 2007년 10월 04일

* University of Nebraska-Lincoln
** 교신저자, 한국정보사회진흥원, (100-775) 서울특별시 중구 무교동 77번지 NIA빌딩

1. Introduction

Extreme Programming (XP) has gained much attention among practitioners and researchers since its successful application to a software development effort at Chrysler [Anderson et al., 1998; Beck, 2000; Jeffries et al., 2001]. The Chrysler Comprehensive Compensation (C3) project, originally launched in May 1997, was declared a failure until the XP methodology was introduced. The project restarted using the XP methodology and was completed with very successful results. In addition to Chrysler's C3 project, success stories regarding Web applications of XP have been reported in recent years [Hieatt and Mee, 2002; Murru et al., 2003].

XP is an agile programming methodology based on the values of simplicity, communication, feedback, and courage. These values are the basis for a number of basic principles which include such things as embracing change, encouraging quality work, and providing rapid feedback [Kendall and Kendall, 2004]. XP is an approach that balances key resources with the need of coding, designing, testing and listening activities.

Pair programming is one of four core practices of XP and actually pairs two programmers for one task. They work together on all aspects and phases of the programming project including analysis, design, implementation, and testing. This approach is believed to spark creativity, reduce errors, and save time [Anderson et al., 1998; Jeffries et al., 2001]. One person, the driver, writes code while the other, the observer contributes to designing effective algorithms or monitors for errors. The roles of driver and observer are usually switched at an appropriate point in time. Compared to a long established practice in which programmers work in isolation, pair programming is a drastic change and an important topic attracting the attention of researchers and practitioners of systems analysis. Today pair programming extends its application to even globally distributed software teams and development practices connected by the Internet or other high-speed networks [Canfora et. al., 2006; Flor, 2006].

Many studies have identified benefits associated with pair programming such as error-free code, greater confidence in work, and high levels of enjoyment [Anderson et al., 1998; Nosek, 1998; Williams et al., 2000]. Even in educating computer science students, the use of pair programming helps paired students to improve tests and projects scores equal to or better than solo students [Williams et al., 2003; McDowell et al., 2006]. Based on the findings, an educational institution examines the feasibility of pair programming as a part of XP method in an attempt to develop a new pedagogical framework for software development methods [Dubinsky and Hazzan, 2005]. These positive effects may accelerate the widespread use of pair programming in development projects. More importantly, it is found that productivity gain is greater in a less experienced pair against a less experience solos than an experienced pair of programmers against a experienced solos [Lui and Chan, 2006], which suggests how organize a project team to take the best advantage of pair programming. According to a recent worldwide survey, 35 percent of 104 development projects reported using pair programming [Chan et al., 1998].

Some studies, however, have found negative effects of pair programming [Hale et al., 2000; Jeffries, 2001; Parrish et al., 2004; Smith et al., 2001]. Recent research showed that pair programming actually wasted time and did not provide any productivity impacts in an industrial setting. Some even argue that working independently is more productive. These observations inspired further research examining other factors that might influence pair programming effectiveness such as ways to pair programmers with different levels of expertise, problem complexity, and development environment in a university setting.

The purpose of this study is to compare the effectiveness of pair programming to independent programming under realistic system development project like conditions where all programmers are allowed to freely communicate with others. This paper is organized as follows: Section 2 provides a review of prior research and description of the research model. Section 3 presents the experimental design. Section 4 contains a discussion of the research results. Conclusions are presented in section 5. Finally, a task set and the Entity-Relationship diagram used in this study are presented in appendices.

2. Review of prior research

According to Kraut and Streeter [Kraut and Streeter, 1995], a major cause of software crises, such as calendar or cost overruns, is the problem of coordinating activities in large software development projects. They defined coordination as "individuals' efforts toward achieving common explicitly recognized goals" and "the integration or linking together of different parts of an organization to accomplish a collective set of tasks." Researchers in software engineering have identified many coordination techniques to improve communication efficiency or performance factors [Anderson et al., 1998; Williams and Kessler, 2000]. Kraut and Streeter [1995] empirically found that informal, interpersonal communication was a valuable method of achieving coordination in typical software projects. Cockburn [2000] revealed that two paired developers at the whiteboard achieved better performance in communication efficiency than those using e-mail, telephone, or videotape. Moreover, two programmers working side by side on the same problem [Williams et al., 2000] not only produced quality software, but also felt more confident and enjoyment related to their efforts [Nosek, 1998; Williams et al., 2000].

Debates about the efficiency and effectiveness of pair programming have revolved around software developers and researchers [Jeffries, 2001; Nosek, 1998; Williams and Kessler, 2000]. Those who advocate pair programming argue that pair programming enables developers to improve the quality of software (i.e., error-free code) and decrease the overall development time. Arisholm et al. [2007] further reveal that pair programming practices are also desirable even in performing maintenance tasks. This yields an increase of productivity in system maintenance as well as system development projects. Furthermore, the members involved in a team usually develop a higher level of confidence in their solutions and a greater sense of enjoyment through collaborative work than independent programmers [Anderson et al., 1998; Beck, 2000; Jeffries et al., 2001; McCormick, 2001; Williams and Kessler, 2000]. Those who are skeptical of the value of pair programming, however, argue that it wastes time for people to work together on the same algorithm or program, and programmers are accustomed to writing code in isolation [Jeffries, 2001]. Industrial field studies found that the more time team members spend concurrently on the same code modules, the less productive they are. In other words, programmers working independently tend to be more productive [Hale et al., 2000; Parrish et al., 2004; Smith et al., 2001].

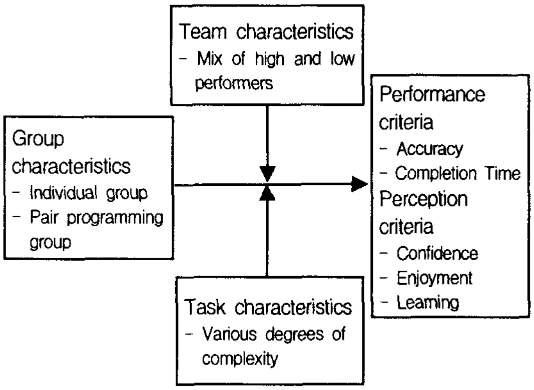
Pair programming was proven to be efficient, both empirically and anecdotally in the C3 software development project [Anderson et al. 1998] and in laboratory experiments using students [Williams et al., 2000] or full-time system programmers [Nosek, 1998; Arisholm et al. 2007]. Nosek [1988] conducted two experiments using different subjects, students and professionals. The subjects consisted of 15 full-time system engineers, with 5 persons assigned to the control group and 5 pairs assigned to the experimental group. The subjects in both groups were asked to code a data integrity check using Sybase, which requires very skilled engineers in C programming to solve the problem. The results of the experiment revealed that the pair-programming group achieved higher quality of functionality, more confidence, and a higher level of enjoyment than the individual group. Some criticize this experiment for not balancing workload between the two groups. They argue that by assigning the same job to both groups, members of the control group had to do twice as much work as members of the paired-group.

To eliminate the workload effect, Williams and his colleagues [Williams et al., 2000] performed a similar experiment, but with a balanced workload between the groups. In this experiment, 41 students participated : 13 subjects were assigned to the control group and 28 students were paired for the experimental group. In addition, unlike Nosek's random assignment methods for the experiment group, Williams et al., classified the students into high, average, and low performers and created teams with a mix of performance levels. This assignment approach for the project team might reflect the way software development project teams are formed in practice. The results of this experiment were similar to Nosek's : the paired-group showed superior results.

Having found that working independently is more productive in a field study [Hale et al., 2000; Smith et al., 2001; Parrish et al., 2004] conducted an industrial field study to examine the productivity effects of pair programming, introducing a concept of concurrency metrics. The concurrency metric refers to the degree to which different programmers report working on the same module during the same day. Pair programmers exhibit very high concurrency, working collaboratively at the same time on the same task. On the other hand, in a low concurrency situation, team members may work together but not on the same day. Software development is a process of collaboration where programmers do not work in total isolation without any opportunity to discuss with or seek help from colleagues. Generally, software development projects are characterized by low-concurrency. Their findings showed that the high-concurrency teams are dramatically less productive than the low-concurrency teams. This result is completely opposite to the findings of previous studies.

What make their results opposite? Through a careful analysis of the research, we recognized there were differences in the level of concurrency among groups (especially the control group). The studies that found positive effects of pair programming tended to arrange the control group in such a way that individuals were not allowed to communicate with or seek help from others. However, in the industrial field study, individuals on a team were allowed to talk to and get help from other team members even if they were not formed as pair programmers. In other words, they were in a low-concurrency situation. We argue that the effects of pair programming will vary depending on the level of concurrency allowed for the control group.

The purpose of this study is to again compare productivity and perceptional variables between pair programmers and individual programmers, however the level of concurrency was also considered an explanatory variable in the experimental design and analysis. The research model is presented in <Figure 1>.



〈Figure 1〉 Research model

3. Research methodology

A laboratory experiment was conducted to determine the effects of pair programming on project performance and participants' perceptions of enjoyment, confidence, and learning. Project performance was measured with accuracy and task completion time. Perceptions were measured with a questionnaire after the experiment. Unlike earlier studies, members of the control group (independent programmers) were allowed communication with others in the group. This established a low-concurrency situation and enabled the analysis of differences in the outcome variables with respect to level of concurrency. The experimental group experiences high-concurrency.

Another major difference of this experiment is that the participants actually executed a programming task, as opposed to only formulating queries. This element is important because programming, usually involves construction of logic and writing and debugging code [Chan et al., 1998]. We believe most coordination problems and the need for help from other team members usually involve the debugging work. One computer loaded with the Oracle client program (SQLPLUS 8.0) was assigned to each pair of the experiment group and to each individual in the control group. Finally, the output files for each group were saved in a network drive for future data analysis.

3.1 Subjects

Seventy-six students, consisting mostly of senior MIS majors enrolled in a database course, participated in the experiment. The average age of the subjects was 22.5, with 49 male and 27 were female students. The subjects held some programming skills such as Visual Basic .Net or Java and were knowledgeable on the basic concepts of Oracle SQL and relational databases.

The subjects were randomly assigned to one of the two groups: the control group or the experimental group. Then the subjects of both groups were paired by matching a low-skilled student with a high-skilled student. Skill level was determined by their performance on earlier class assignments and their grade point average. This approach helped balance the skill level of each team, which is more similar to real-world software development environments where junior programmer are often paired with more senior developers.

In the control group, participants were paired as above and were allowed to discuss their project with their partner; however, they were each required to solve the problems independently and accomplish the work individually at separate work stations. The results from each member of the control group were evaluated individually, while the results from the members of the experimental group were submitted by true pairs and evaluated as a team. All participants were offered extra credit based on their performance.

3.2 Task characteristics

A booklet provided instructions for a set of tasks representing three complexity levels (easy, medium, and hard). A single table, multiple tables, and creation of a nested query were the criteria for task complexity. To provide equal workloads for both groups, the pair programming group was required to solve two problems for each level, while individuals in the control groups were assigned only one problem per level.

Both groups were presented with the same Entity-Relationship (ER) diagram, consisting of eight tables and their relationships and attributes, as a conceptual data model (see Appendix A). Before the experiment, the ER diagram was converted into a physical data model and then its tables and constraints were created in the Oracle Database Management System (Version 8.0).

3.3 Procedures

Two separate computer labs were reserved for the study. Both labs were equipped with 38 Windows XP based PCs loaded with the Oracle client program (SQLPLUS 8.0). All subjects were familiar with the labs because they were used for the database course. Before the experiment began, the rules and the guidelines were explained to all participants. Time limits of 8, 10, and 12 minutes were assigned to easy, medium, and hard-level tasks, respectively (see Appendix B). Participants completing a task before the time limit were instructed not to start the next task until the scheduled time. The subjects were instructed to record their start and end time for each question. The students used SQLPLUS to formulate, and run their queries. SQLPLUS is an interactive system, with a DOS-based user interface. If an error occurs, SQLPLUS indicates the nature of the problem. Since the students had learned basic SQL syntax for retrieval and had more than three Lab sessions, they were not allowed to refer to class notes, the SQL manual, or the textbooks in this experiment.

3.4 Dependent variables

The dependent variable performance was measured by task accuracy and completion time of the queries. The perceived degree of enjoyment from collaborative work with the partner, confidence in results, and learning from the partner were measured with an exit survey questionnaire. The questions on the construct of enjoyment were drawn from previous research [Agarwal and Karahanna, 2000]. The scale used for the dependent variables of subject perceptions ranged from 0 to 7 (0 = "strongly disagree," 7 = "strongly agree").

4. Results

T-tests and two-way ANOVA were used for data analysis. Details of the dependent variables-accuracy, task completion time and perceptions on enjoyment, confidence, and learning are provided in the following sections.

4.1 Accuracy

Before grading a subject's answers, a grading criterion was chosen based on critical clauses of SQL syntax to be included in the answer for each problem. Two instructors independently determined the accuracy of the answers, based on the grading criterion. The points for each answer ranged from the minimum of 0 to the maximum of 5. The correlation coefficient of the grades from the two instructors was 0.91, showing a high inter-rater reliability for the measure of accuracy. The total score of pair programming teams in the experiment group was compared to the total score of each two-person teams in the control group. The highest score possible for the tasks is 30 points as there were six questions with five maximum points each.

The accuracy mean and standard deviation for each groups is shown in <Table 1>. As expected, the mean scores for the two groups are similar and accuracy between the two groups is not statistically different (t-value =1.64, p =.12).

〈Table 1〉 Performance accuracy of the groups

Group	N (team)	Mean	Std. Deviation
Pair Programming	19	23.187	3.654
Individual	19	25.750	2.478

Subsequently, we analyzed the task performance with respect to task complexity for each group. The mean accuracy and standard deviation for each complexity level are summarized by group in <Table 2>. We found that the individual (control) group outperformed the pair programming group for the hard-level task ($F = 6.86$, p-value < 0.05) and there were no significant differences between groups for the easy and medium-level tasks ($F = .07$, p-value = .80 ; $F = .05$, p-value = .83).

〈Table 2〉 Performance accuracy and task complexity

Task complexity	Group	N (team)	Mean	Std. Deviation
Easy-Level	Pair Programming	19	9.500	1.069
	Individual	19	9.312	1.751
	Total	38	9.406	1.404
Medium-Level	Pair Programming	19	8.875	1.356
	Individual	19	8.750	.886
	Total	38	8.812	1.108
Hard-Level	Pair Programming	19	4.812	2.658
	Individual	19	7.687	1.602
	Total	38	6.250	2.588

It should be noted that these results are contrary to Nosek's results [Anderson et al., 1998], which showed that the pair programming group outperformed the individual group.

4.2 Completion time

Since all tasks were not completed by all participants, completion times in the analysis reflect only completed tasks. The means and standard deviations for each question and groups are shown in <Table 3>. There is no significant difference in the task completion time between the groups. For the last problem (Q6), since no team in the pair programming group had a correct answer, a comparison between groups was excluded.

〈Table 3〉 Measures of elapsed times for questions for the groups

	GROUP	N (team)	Mean (Minutes)	Std. Deviation (Minutes)	t-value	p-value
Time for Q1	Pair Program	18	1.714	.487		
	Individual	17	2.000	1.154	.60	.558
Time for Q2	Pair Program	15	1.833	1.329		
	Individual	17	2.571	1.511	.93	.37
Time for Q3	Pair Program	18	2.714	1.112		
	Individual	14	2.600	1.516	-.15	.88
Time for Q4	Pair Program	12	3.750	1.707		
	Individual	14	4.200	3.271	.25	.81
Time for Q5	Pair Program	14	3.800	1.923		
	Individual	8	1.666	.577	1.82	.12

4.3 Subject perceptions

As mentioned earlier, enjoyment of collaborative work with a partner, confidence in the results, and learning from a partner were assessed via an exit survey questionnaire. The reliability measures of enjoyment, confidence, and learning based on Cronbach's alpha were 0.82, 0.89, and 0.78, respectively. Only learning from the partner in the individual (control) group was significantly different from the pair programming group at the .10 level ($F = 3.42$, $p\text{-value} = 0.07$). The means and standard deviations for the three perception variables of the two groups are shown in Table 4, and we found that the perception of learning is higher in the control group than the pair programming group.

〈Table 4〉 Analysis of perception

	group	Mean	Std. Deviation	N	F	p-value
Enjoyment	Pair Program	5.708	.909	38	1.34	.26
	Individual	6.062	.818	38		
	Total	5.885	.870	76		
Confidence	Pair Program	3.968	1.892	38	2.28	.14
	Individual	5.000	1.966	38		
	Total	4.484	1.969	76		
Learning	Pair Program	5.145	1.192	38	3.42	.07
	Individual	5.895	1.100	38		
	Total	5.520	1.191	76		

5. Discussion

The results indicate that there is no significant difference between the experiment group (pair programming) and the control group (individuals) when the control group is exposed to an environment very similar to the real-world system development setting. Interestingly, the individual group outperformed the pair programming group for the hard task, while there were no differences for the easy and medium level problems.

None of the other measures, such as completion time, enjoyment, or confidence, was significantly different between the two groups. When the subjects in the control group worked individually in the collaborative work environment, they experienced as much enjoyment as pair programmers did. We can infer from this result that team members in software development projects would feel enjoyment of collaboration in itself. Even though the individual group had higher scores for the hard problems, the subjects in the group did not show any difference in their confidence level. This result could be due to the fact that both groups executed queries using SQLPLUS. In other words, they could see their answers immediately after executing queries even if the answers were wrong, which might have caused their confidence levels to be similar.

For the perception of learning from the partner, the individual group again showed a higher level than the pair programming group. We believe the reason for this result could be based on the fact that each individual sought his/her partner's help after the initial effort. We observed that each subject in the individual group began talking soon after the starting time to solve each problem, which lead them to ask or discuss only about their concerns. The critical help from their partners to alleviate or remove their initial concerns, in turn, might have made the individuals to perceive a higher level of learning from their partners.

The results of this study provide some new insights regarding pair programming. First, our results indicate that in the communication-allowed environment, which reflects more accurately the real-world system development environment, pair programming is not superior to the traditional programming development approach. Here, we are not suggesting that the traditional method is better than pair programming either. At least in an apples-to-apples comparison situation, however, there was no significant difference. Pair programming scholars argue that the real value of the approach is the ability to identify early design or coding problems, which later could save development time and costs. We do agree that for collaborative efforts, such as error checking, that assertion could be true. However, we also agree with the critics of pair programming who argue that working together on the same algorithm or programming maybe a waste of time [Jeffries, 2001].

Second, our results indicate that, depending on task complexity, pair programming might be less effective than the traditional approach. We did not see any difference for the easy and medium level tasks, but the individual group showed higher performance for more difficult tasks than the pair programming group. Although it is difficult to generalize the results based on one experiment, there are signs that individuals might perform better for more creative tasks, and not-well-coordinated groups might even out their capability when teamed with inferior performers. We should investigate with care as to when and under what conditions one approach would do better than the other.

This experiment has several areas that need further improvements. First, an exercise or training session before the experiment might be needed to alleviate unfamiliarity between team members. We observed that discussion about the given problems occurred only after some time had elapsed after the experiment started. Since all the subjects in the experiment were classmates, we thought they already knew each other sufficiently enough to alleviate any unfamiliarity provoked by sudden team matching. Since in real-world cases, some teams are formed without having prior familiarity, we do not think this arrangement seriously affected our results. However, we conjecture that an early team-building exercise might facilitate more collaboration among the team members. Second, we believe the sequence of task complexity should be randomized. In our experiments we asked the subjects to solve problems in the sequence of easy, medium and hard tasks. Since learning is an important factor in performance, the sequence might be interplayed with the learning. Finally, the employed tasks may not be considered as a close replication of what most programmers carry out routinely. In fact, SQL is relatively simple in terms of reflecting programmer's creativity and effectiveness of their performance at work. Despite the loss of generalizability, we believe task of SQL query enable us to control another intervening factor, task complexity at three different levels which is rarely manipulable in real-world situations.

6. Conclusion

In this paper we examined the effects of pair programming on task performance in an experimental setting that is close to the software development environment in practice. The results of this study supported our initial position that there would be no significant difference in task performance between the experiment group (pair programmers) and the control group (individuals working independently but allowed to communicate with each other). This result is contrary to the findings of previous research which revealed that pair programming groups achieved higher quality of functionality, more confidence, and more enjoyment than traditional groups [Nosek, 1998; Williams et al., 2000].

Different results based on different experimental design environments warrant more investigation into this issue. In particular, we believe that the mix of teams, existence of collaboration mechanisms, the degree of task complexity and types of work (whether it is a design or programming task) will play important roles in determining the merit of pair programming.

References

[1] Agarwal, R. and Karahanna, E., "Time flies when you're having fun : cognitive absorption and beliefs about information technology usage", *MIS Quarterly*, Vol. 24 No. 4, 2000, pp. 665-694.

[2] Anderson, A., Beattie, R. and Beck, K., "Chrysler goes to Extreme", *Distributed Computing*, 1998, pp. 24-28, available at http://www.xprogramming.com/publications/distributed_computing.htm.

[3] Arisholm, E., Gallis, H., Dyba, T., and Sjoberg, D. I. K., "Evaluation pair programming with respect to system complexity and program expertise", *IEEE Transactions on Software Engineering*, Vol. 33 No. 2, 2007, pp. 65-86.

[4] Beck, K., *Extreme Programming Explained*, Addison-Wesley, Upper Saddle River, NJ, 2000.

[5] Canfora, G., Cimitile, A., Di Lucca, G. A., and Visaggio, C. A., "How distribution affects the success of pair programming", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 16, No. 2, 2006, pp. 293-313.

[6] Chan, H., Siau, K. and Wei, K., "The effect of data model, system and task characteristics on user query performance - an empirical study", *The Data Base for Advances in Information Systems*, Vol. 29, No. 1, 1998, pp. 31-46.

[7] Cockburn, A., "Selecting a project's methodology", *IEEE Software*, Vol. 17, No. 4, pp. 64-71.

[8] Cusumano, M., MacCormack, A., Kemerer, C. and Crandall, B. (2003), "Software development worldwide: the state of the practice", *IEEE Software*, Vol. 20 No. 6, 2000, pp. 28-34.

[9] Dubinsky, Y. and Hazzan, O., "A framework for teaching software development methods", *Computer Science Education*,

Vol. 15 No. 4, 2005, pp. 275-296.

[10] Flor, N., "Golbally distributed software development and pair programming", *Communications of the ACM*, Vol. 49 No. 10, 2006, pp. 57-58.

[11] Hale, J., Parrish, A., Dixon, B. and Smith, R., "Enhancing the Cocomo estimation models", *IEEE Software*, Vol. 17, No. 6, 2000, pp. 45-49.

[12] Hieatt, E. and Mee, R., "Going faster: testing the Web application", *IEEE Software*, Vol. 19, No. 2, 2002, pp. 60-65.

[13] Jeffries, R., Anderson, A. and Hendrickson, C., *Extreme Programming Installed*, Addison-Wesley, Upper Saddle River, NJ, 2001.

[14] Jeffries, R., "What is extreme programming?", 2001, available at : http://www.xprogramming.com/(accessed October 2005).

[15] Kendall, K. and Kendall, J., *Systems Analysis and Design*, 6th Edition, Pearson/Prentice Hall, Upper Saddle River, NJ, 2004.

[16] Kraut, R. and Streeter, L., "Coordination in software development", *Communications of the ACM*, Vol. 38, No. 3, 1995, pp. 69-81.

[17] Lui, K. M. and Chan, K. C. C., "Pair programming productivity : Novice-novice vs. expert-expert", *International Journal of Human-Computer Studies*, Vol. 64, 2006, pp. 915-925.

[18] McCormick, M., "Programming extremism", *Communications of the ACM*, Vol. 44, No. 6, 2001, pp. 109-111.

[19] McDowell, C., Werner, L., Bullock, H.E. and Fernald, J., "Pair programming improves student retention, confidence, and program quality", *Communications of the ACM*, Vol. 49, No. 8, 2006, pp. 90-95.

[20] Murru, O., Deias, R. and Mugheddu, G., "Assessing XP at a European Internet company", *IEEE Software*, Vol. 20, No. 3, 2003, pp. 37-43.

[21] Nosek, J., "The case for collaborative programming", *Communications of the ACM*, Vol. 41, No. 3, 1998, pp. 105-108.

[22] Parrish, A., Smith R., Hale, D. and Hale, J., "A field study of developer pairs: productivity impacts and implications," *IEEE Software*, Vol. 21, No. 5, 2004, pp. 76-79.

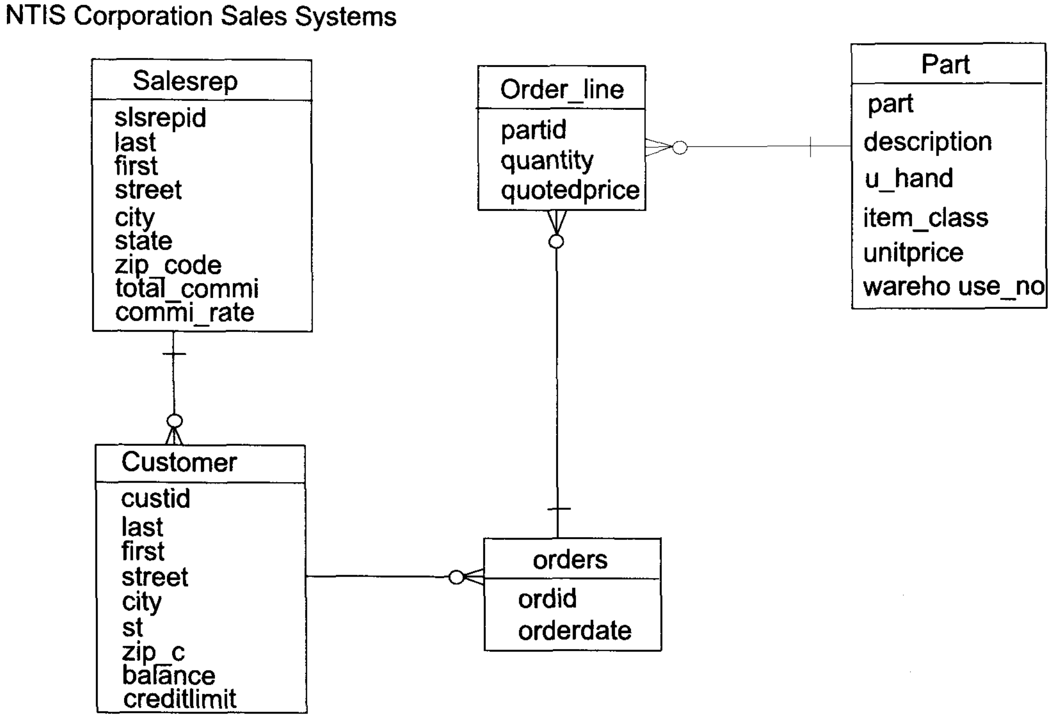
[23] Smith, R., Hale, J. and Parrish, A., "An empirical study using task assignment patterns to improve the accuracy of software effort estimation", *IEEE Transactions on Software Engineering*, Vol. 27, No. 3, 2001, pp. 264-271.

[24] Williams, L. and Kessler, R., "All I really need to know about pair programming I learned in kindergarten", *Communications of the ACM*, Vol. 43, No. 5, 2000, pp. 108-114.

[25] Williams, L., Kessler, R., Cunningham, W. and Jeffries, R., "Strengthening the case for pair-programming", *IEEE Software*, Vol. 17, No. 4, 2000, pp. 19-25.

[26] Williams, L., McDowell, C., Nagappan, N., Fernald, J. and Werner, L., "Building pair programming knowledge through a family of experiments", *Proceeding of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)*, 2003, pp. 143-153.

Appendix A. The conceptual model(ER Diagram) and sample data



Appendix B. The tasks set

Difficulty level		Question contents
Easy	Question 1	Find out how many customers have a balance that is less than their credit limit.
	Question 2	List customer last name (LAST), Balance, and credit limit in ascending order of their last name.
Medium	Question 3	List PARTID, QUANTITY, and QUOTEDPRICE of parts that were sold on July 4th, 2001.
	Question 4	List the sum of balance of all customers for every sales rep. Order the results using the sales rep number.
Hard	Question 5	List CUSTID, FIRST, LAST, and BALANCE for every customers whose BALANCE is greater than average balance.
	Question 6	Calculate each sales rep's total commission for July sales. (Hint : commission = Quantity*Quotedprice*each sales rep's commission rate (COMMI_RATE))

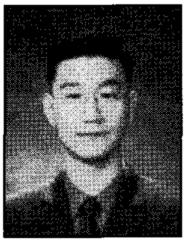
■ 저자소개

윤 성 노

아주대 전산학 학사, University of Nebraska-Lincon에서 전산학 석사를 취득하였으며 현재 동대학 경영정보학 박사과정에 재학중이다. LG-CNS와 포스데이타에서 시스템개발 및 사이베이스 등 실무경험을 쌓았다. Computer Information Systems, Industrial Management & Data Systems, Management Science 등에 다수의 저널을 게재하였으며 주요 관심분야는 전자상거래, 정보시스템 개발방법론, 유비쿼터스 컴퓨팅 등이다.

김 종 헌

University of Nebraska-Lincoln에서 MBA를 취득하였으며 현재 동대학 경영정보학 박사과정에 재학중이다. 최근 연구분야는 새로운 미디어의 효용성, 정보윤리와 정보보호, 그리고 virtual presence와 physical presence의 상호연관성 등이다. Journal of CIS와 IJITM 등에 다수의 저널을 게재하였다.



박 상 현

한국항공대에서 항공경영을 전공하고 충북대학교에서 경영정보학을 전공하여 석사와 박사를 취득하였으며 University of Nebraska-Lincoln에서 박사후과정을 수료하였다. ㈜솔리데오시스템즈, 한국전자통신연구원(ETRI), 정보통신정책연구원(KISDI) 등 IT 전문 기업 및 연구기관에서 근무하였으며 현재 한국정보사회진흥원(NIA)에서 선임연구원으로 재직하고 있다. Technological Forecasting & Social Change, Industrial Management & Data Systems, Telecomunications Review 등의 저널에 다수의 논문을 게재하였으며 주요 관심분야는 유비쿼터스 컴퓨팅, 디지털 컨버전스, 정보통신 정책, 국가정보화전략, 지식공유와 u-미디어 등이다.

◆ 이 논문은 2007년 08월 24일 접수하여 1차 수정을 거쳐 2007년 10월 04일 게재확정되었습니다.