2011 International Conference on Humanities, Society and Culture
IPEDR Vol.20 (2011) © (2011) IACSIT Press, Singapore

Problem Decomposition Technique and Cooperative Learning in Enriching Software Development Projects

Tie Hui Hui [1] and Irfan Naufal Umar [2 +]

[1] Centre for Postgraduate Studies, SEGi College Penang, Malaysia
[2] Centre for Instructional Technology & Multimedia, University Science Malaysia, Malaysia

Abstract. Problem decomposition as chunking technique is used as abstract concept to break the programming scenarios into smaller manageable chunk in relation to fulfilling the system requirements. The purpose of this study is to investigate the effects of problem decomposition with cooperative learning on developing software in terms of the number of revised project proposal, self-programming and project performances. A total of 44 second year computing students were randomly assigned either to a group that received a combination of problem decomposition and cooperative learning (DCL), or to another group which received cooperative learning method (CL) in this 28-weeks treatment. The participants worked in a group of four, with a mix of students with high and low self-regulated learning levels. The post-assessment was administered to measure their software development performance. The analysis results revealed that students in the DCL group performed significantly better than those in the CL group in both the self-programming performance and software project development. Also, the DCL group made fewer amendments to the project proposal than their counterparts. Thus, the problem decomposition technique incorporated into the "problem and analysis stages" within the system development cycle should be considered as an alternative strategy for effective way of teaching and completing software project.

Keywords: cooperative learning, problem decomposition, system development project

1. Introduction

Information system (IS) has an influential role in assisting organisation to achieve success, gain competitive advance and increase profit. As such, the expectation on ICT projects is increasing. More sophisticated and advanced functionalities with user friendliness features are in demand. Besides technological challenges, business needs, user requirements, organisational expectation issues, experiences and expertise of IS project teams are contributing failure to development initiatives [1]. The IS project development requires managerial, coordination, problem solving, critical thinking and analytical skills. These essential skills ought to be grasped in studying programming courses. On the other hand, programming demands complex cognitive skills that students find them too complicated to understand, interpret and perform. Likewise, educators involved in teaching software development process are continually facing different challenges in cultivating the students' understanding of the systems' concepts, programming activities, development tools and techniques. During the learning process, students need to understand both user requirements and the functionalities of the software. Once these needs have been identified, determining the scope and establishing business requirements are carried out prior to software design, implementation and maintenance. These problem solving, planning, critical thinking, reasoning and social interaction are the skills that computing students find too difficult to acquire. Helwig [2] indicated that understanding the software development process, and learning how, why and when to create deliverable software are through working with "real system development project". The engagement in developing software project could foster growth in problem solving, critical thinking and interpersonal skills. Problem

+ Corresponding author. Tel.: + 6046535230; fax: +6046576749.
E-mail address: (irfan@usm.my).

decomposition as chunking method is a systematic way of decomposing the problem scenario into smaller manageable sub-problems [3]. With its emphasis on problems rather than solutions, this approach allows the students to understand the idealised problems and then to link them with their specific domain knowledge in order to drive the requirement and analysis engineering process. Poveda and Borras [3] stated that decomposition is a way of analysing and managing complexity. It supports both the high-level abstract view of requirements and the lower-level detailed view of processes. As such, could this approach assist the students in identifying the novel scenario and then in decomposing the scenario into sub-scenarios by looking at all the possible development constraints?

2. Problem Statement

In programme development, it is not just about identifying the basic requirements stated in the given novel scenarios; and it is also not merely about learning some programming language syntax. It requires the abilities to handle the software development process and to turn the problem scenario into a valid workable application. Empirical research findings disclosed that undergraduate computing students with limited theoretical programming knowledge, problem solving and practical skills are facing challenges in establishing project objectives, defining software functionalities and completing the system (software) within the given timeline [4]. For successful software development, these knowledge and practical skills are needed to be applied in all phases of development cycle which students seem to lack. Likewise, the initial findings in Table 1 reveal that the problems faced by these students are in accordance with literature concerning problems in computing programming. The preliminary statistical results indicated that 20 percent of the students' project scope had been revised after their project proposal submission; only few students had completed the system on time and achieved the project objectives. In line with the findings reported by [5] on software projects failure in the software industry that achievable project objectives, executable functionalities and fulfilling the users' requirements are the key to IT project success.

Table 1: Problem in developing software engineering.

Requirement and analysis stage	
• scopes of the system are too large or too visionary	• ill-defined functional modules
• lack of clearer software scope	• poor planning and time management
• uncertain with the project objectives	• incomplete testing
• lack of stakeholder participation in the design	• limitation on programming language knowledge
• literature review is too brief or incomplete	• incompatibility between technology and programming languages
• lack of problem solving, analytical and reasoning skills	
• lack of programming conceptual understanding	

Programme development begins with problem identification. Once this problem requirement has been identified, the selected programming languages will be applied in the implementation stage. Planning, logical reasoning, problem solving and critical thinking are the skills required in the process of learning and foremost during software development [6]. In order to achieve higher success rate in software development, these knowledge and skills are crucial and needed to be applied throughout the software development phases, which the students are facing difficulty to acquire them. Using the combination of programming and problem decomposition technique effectively in course delivery, logical reasoning and problem solving skills can be cultivated while learning to develop a valid workable application [6].

The preliminary questions asked prior to the development process are based on the nine topic areas such as (i) business scope of organisation, (ii) business requirements, (iii) user requirement, (iv) system requirements, (v) project scope, (vi) project timeline pressure, (vii) technology compatibility, (viii) functionalities / technical complexity, and (ix) programming language. With lots of uncertainties at the preliminary stage, there is no surprise that the project scope, requirement and constraints (technical and programming tools) are frequently revised throughout the developing cycle. This gives the students insufficient time to implement the specifications as stated in the software proposal; and may not have sufficient duration for evaluating the product [7]. Despite this initial investigation and involvement, software development constraints in relation to external factors and development factors should be emerged. By looking at these constraints at the early stage of the development, it will help the students to select the suitable development methodology, approaches and design for their new project. Also, it encourages the

students to analyse the success rate of this project as well as the potential benefits of such implementation. This further assists in strengthening the understanding of the new knowledge acquired. These entire activities indicate active, reflective exploratory and experimental processes of the learning approach.

The five main stages in the software development process begins with problem analysis, then proceeds to planning the solution, coding, testing and ends with the documentation. It requires the students to understand the programming process and to incorporate these process stages into the system development methodologies. Likewise, the stakeholders' requirements and system objectives are crucial to the success of IS development project. As business system and environment undergo continual changes, the IS should keep pace. Somehow this revolution has contributed to the constant changes in business and users' requirements. However, it is a challenge to determine the scope of the system and to establish the software requirements as users do not always convey their needs and expectations. This could be the fact that they may not fully understand precisely what are needed in the first place. With such uncertainties, students find the development process time consuming and tedious as they need longer time to have solid understanding of the users' requirements and the system's functionalities. Adding to the challenge is when the students discover the incompatibility of the technology device with programming languages at the coding stage. Also, the literature review of such similar product was usually not done yet at the beginning of the IS development process until when necessary. By then the project scope has been firmed up and final user requirements are established and any modification or additions to the project proposal will contribute to the delay in the project completion.

3. Research Methodology

The purpose of this study is to investigate the effects of problem decomposition with cooperative learning (DCL) and cooperative learning only (CL) methods on computing students in developing final product (software). It aims to examine whether the combination of problem decomposition techniques and cooperative learning is an effective solution in programming education during software development.

The assessment of students' programming development performance includes: (i) number of revisions to project proposal, (2) self-programming performance, and (3) software project performance. As such, their performance was measured based on the scores obtained from the number of revised project proposal, the self-programming appraisal (SPA) questionnaire, and software project evaluation (SPE) mark sheet. The *Motivated Strategies for Learning Questionnaire* consists of 23 items was used to identify the students' self-regulated learning level (high or low SRL) prior to the treatment. In this pilot study, a quasi-experimental design was used that involved 44 second year computing course students. These two intact group classes were randomly assigned to the two treatment groups. The experimental group (20 students) received the DCL treatment while the control group (24 students) received the CL treatment. Those in the DCL group received the combination of problem decomposition and cooperative learning strategy in acquiring the basic software development methodologies. In the CL group, they were only exposed to cooperative learning strategy in developing the group project. To understand the concepts of software engineering, both groups were taught to analyze the types of software process techniques (e.g:. spiral methodology, rapid prototyping model, incremental model, object oriented programming, and Agile software development).

In this study, the course comprises lectures and practical sessions. The students in both experimental groups worked cooperatively in a team of four members. Each team consisted of two high and two low SRL students. During the second week of treatment, the students had to submit their project proposal and the number of changes made after the first submission was recorded. They were persistently required to cooperate on every programming activity throughout the development stages. Once the coding has been completed by the individual team member, the integration of all modules was carried out to ensure the compatibilities of the functions within the product (software). In the second semester, they concentrated on developing the software and the role of lecturer has been switched to facilitator mode. It is to ensure that the students achieve all objectives and complete the project within stipulated timeframe. In Week 28, both groups were assigned a presentation slot to demonstrate their software. Immediately after the treatment, each group was given an hour to demonstrate their software. Immediately after their software presentation, the students were asked to complete the SPA questionnaire to measure their self-progression on the programming knowledge and skills gained throughout the development cycle. Meanwhile, the SPE

instrument was used to measure the students' logical understanding of the software codes, strategic/conditional knowledge and programming skills. These two instruments were used to assess the students' programming performance in terms of coding skills, programming knowledge and software process techniques. Also, the number of revisions made to the project proposal was measured as to determine their understanding on the initial problems, user requirements and scope of the system.

4. Research Findings

MANOVA statistical technique was applied to test the research hypotheses on the three dependent variables: (i) number of revisions to proposal, (ii) self-programming performance and (iii) software development performance. The MANOVA results clearly revealed significant differences in all the three dependent variables stated in the research hypotheses.

Hypothesis 1: There was no significant difference in the number of revisions made to the project proposal between the DCL and CL groups. In this study, the MANOVA analysis result indicated a significant difference in the number of revisions made to the proposal between the DCL and the CL group (F: 13.62; p: 0.001), with the DCL group performed significantly less amendments to the project scope in the proposal than the CL group ($\overline{X}_{DCL}$: 1.40; $\overline{X}_{CL}$: 2.75). Therefore, this finding has rejected the first hypothesis.

Hypothesis 2: There was no significant difference in self-programming performance between the DCL and CL groups. The result indicated that there is a significant difference in self-programming performance between both groups (F: 13.10; p: 0.001). The students who received DCL treatment significantly outperformed those of CL treatment ($\overline{X}_{DCL}$: 3.71; $\overline{X}_{CL}$: 3.26) in self programming knowledge assessment. Thus, the second hypothesis has been rejected.

Hypothesis 3: There was no significant difference in the project performance between the DCL and CL groups. The finding indicated a significant difference in the software development performance between the DCL group and the CL group, with the former performing significantly better than the latter group (F: 14.68; p: 0.000; $\overline{X}_{DCL}$: 77.60; $\overline{X}_{CL}$: 69.67). Thus, this finding has also rejected the third hypothesis.

5. Discussion

The research findings revealed a significant difference between students taught in the two instructional methods on all the three dependent variables. The analyses indicated that students in the DCL group made significantly less adjustments to the project proposal than their peers in the DL group. The DCL group had also shown significant result on the capability of self evaluation in terms of their self-programming performance as compared to the CL group. Similarly, the DCL group significantly outperformed the CL group on project development performance. As such, the DCL method significantly influenced the students' programming performance as they progressed.

The problem decomposition with cooperative learning method significantly assisted the students in identifying the potential requirements and risks by zooming at all possible angles while determining the software scope. This technique helped the students to focus on the novel problems and break them into smaller manageable chunks for further analysis. This finding is in line with [3] and [8] that problem decomposition technique as chunking method did assist the students taught in the DCL group in dividing the tasks into syntactically related non-overlapping groups of requirements (users or system). In accordance with [2], this micro level of identifying and analysing problems at each IS development process allows the DCL students to visualise software requirements from the users' perspective. Based on user requirements, the evaluation on business processes, realistic expectation and constraints on software project from the macro level to micro level have helped them in creating a pipeline of new challenging tasks for the software project. This allows the students to identify suitable and manageable project scope so as to complete the project within the defined timeline. In other words, the recognition of requirement, resources and constraints, as well as the discovering of problems are crucial to software project success. Through problem decomposition approach, it encourages the students to look at possible components and analyse every possibility, thus improving analytical thinking and problem solving skills. Therefore this finding demonstrated that students who worked on decomposition through problem-based learning had invested more effort and time on

requirements investigation and analysis and shown deeper understanding on software components as compared to those taught in the CL method. In turn, they have demonstrated better ability to complete the software project within the period of time. Likewise, students in the DCL group made significantly fewer changes to the project scope in their project proposal. This problem identification process applied in software development stages allowed students to set reasonable challenging project scope that is within their capabilities for completion without overly ambitious in building such fully operational system.

This problem decomposition technique allows students to brainstorm on the novel problems that stimulates higher cognitive thinking and in turn cultivates "self troubleshooting" abilities [3]. Through team interaction, the discussions helped each member to ask "why" and "how" on programming statement that somehow reduce mistakes, increase quality in codes and shorter development cycles as well as increase self confident level. This further increases the software project success rate.

6. Conclusion

The study has emphasised the importance of problem decomposition and cooperative learning technique in learning software development. This technique allows students to chunk problem scenarios and user requirements into manageable functions. With this, students have deeper understanding of initial problems and requirements. It emphasizes on problem solving through decomposition, system requirements, application development and implementation, which in turn have influenced students' programming performance. Likewise, it enables the students to cultivate analytical, logical and problem solving skills in the software project environment. Thus, this strategy helped to close the gap between both stages in the practical step and general research analysis principles and to reduce the software project failure rate. Thus, this technique could be of great value in terms of guiding the students to identify, understand and make prompt decisions throughout the project development process. Subsequently, it promotes the development of programming knowledge and competency within self through breaking abstract scenarios into smaller chunks. In teaching and learning, educators should promote and enforce problem decomposition technique throughout the development activities that in turn will cultivate innovative software-building and reveal higher programming achievement.

7. References

[1] B. Hughes, and M. Cotterell. Software project management. Maidenhead: McGraw Hill. 2006.

[2] J. Helwig. Using a "real" systems development project to enrich a systems analysis and design. Proc. of ISE Conference. Columbus. 2005, pp. 1-6.

[3] J. P. Poveda, and J. T. Borras. Inductive logic programming and its application to the temporal expression chunking problem. Ph.D. Programme on Artificial Intelligence (UPC). LSI Department Technical Report – January. 2007.

[4] S. Shahida, T. K. Ahmad, Z. Zurinahni, and S. Sarina. System development: What, why, when and how CASE tools should support novice software engineering. The 3rd Malaysian Software Engineering Conference. 2007, pp. 256-260.

[5] L. A. Kappelman, R. McKeeman, and L. Zhang. Early warning signs of IT project failure: The dominant dozen. Information Management Journal. 2006, pp. 31-36. Retrieved May 2, 2011, from http://www.ism-journal.com/ITToday/projectfailure.pdf

[6] I. Miliszewska, and G. Tan. Befriending computer programming: A proposed approach to teaching introductory programming. Issues in Informing Science and Information Technology. 2007, 4: 277–289.

[7] R. Spencer. The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. CHI Letters. 2000, 2(1): 353-359.

[8] K. Taku, and M. Yuji. Use of support vector learning for chunk identification. In Proceeding of CONLL-2000 and LLL-2000. 2000, pp. 142-144.