Problem Decomposition and Cooperative Learning: an Exploratory Study in Enhancing Software Engineering Projects

Tie Hui Hui and Irfan Naufal Umar

***Abstract*—Completing workable software within stipulated timeframe has always been a challenge for computing students. College educators teaching programming courses are facing difficulties to guide and monitor the students from the start of the software development cycle until the completion stage. Likewise, students find the software development process much more tedious, complicated and frustrating. 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. This paper presents an overview of the difficulties encountered, and a study on problem decomposition technique with cooperative learning to resolve such affliction. 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-week 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 that was based on self-programming appraisal, project performance and number of proposal revision. The 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, learning and completing software engineering project.**

***Index Terms*—Chunking technique, cooperative learning, problem decomposition, software development project.**

I. Introduction

With the advancement of technology, organizations are constantly incorporating utilizing the Information system (IS) in their daily business processes in order to achieve success, stay competitive, increase profit and market share. With this, the expectation on information communication technology (ICT) projects is increasing. More sophisticated and advanced functionalities with user friendliness features are in demand. In addition to technological challenges, business needs, user requirements, organizational expectation issues arise, experiences and expertise of IS project teams are contributing failure to an extensive number of development initiatives [1]. The development of IS project requires managerial, coordination, problem solving, critical thinking and analytical skills. These are the key skills that need to be developed and grasped in studying programming courses. Conversely, programming demands complex cognitive skills that students find them too complicated to understand, interpret and perform. Likewise, educators involved in teaching the software development process to computing students are continually facing different challenges in delivering the system's concepts, programming activities, development tools and techniques; and students are having difficulties to comprehend them. During the learning process, students are to understand both basic requirements such as (i) the user requirements and (ii) the functionalities of the software to perform those requirements. Once these needs have been identified, determining the scope and establishing business requirements are carried out before the stages of software design, implementation and maintenance. This software development process involved problem solving, planning, critical thinking, reasoning and social interaction that computing students find these combination of skills too difficult to acquire. [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.

Based on problem decomposition technique, students are to look at all the possible requirements stated by the users and then to analyze them. In this case, problem 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 idealized problems and then to link them with their specific domain knowledge in order to drive the requirement and analysis engineering process. Meanwhile, [3] stated that decomposition is a way of analyzing and managing complexity. It is a problem solving technique that to be applied in the requirement and analysis phase for identifying the scope of the system. Subsequently, it is incorporated into the design phase so as to determine the logical structure and functionality for building the system. This approach supports both the high-level abstract view of requirements and the lower-level detailed view of processes.

Thus, 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

Manuscript received December 14, 2011; revised February 3, 2012

H. H. Tie is with the Centre for Postgraduate Studies, SEGi College Penang, Malaysia, 43 Green Hall, 10200 Penang, Malaysia (e-mail: hhtie@segi.edu.my).

I. N. Umar is with the Centre for Instructional Technology & Multimedia, University Science Malaysia, 11800 USM, Penang, Malaysia (e-mail: irfan@usm.my).

constraints? Also, will this approach help the students to turn problems into solutions by creating new ideas?

II. Problem Statement

Software project development is not just about identifying the basic requirements stated in the given novel scenarios. It is also not merely about learning some programming language syntax. This development process requires the abilities to handle the software engineering stages 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 and to be applied in all phases of development cycle which students seem to lack. Likewise, the initial findings in Table I reveal that the problems faced by undergraduate computing students in learning programming and developing systems 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. With this, only few students had completed the system on time and achieved the project objectives. However, majority of them were still working on it until the last minutes with incomplete functionality. Thus, only limited testing was conducted. If project duration is not a factor, then a comprehensive testing strategy could be implemented and undiscovered errors may perhaps be detected. This will somehow increase the robustness of the implemented software. 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 I: Problem in Developing Software Engineering

Requirement and analysis stage
• scopes of the system are too large or too visionary
• lack of clearer software scope
• uncertain with the project objectives
• lack of stakeholder participation in the design
• literature review is too brief or incomplete
• ill-defined functional modules
• poor planning and time management
• incomplete testing
• limitation on programming language knowledge
• 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 these problem requirements have been identified, the selected programming languages will be applied in the implementation stage. Programming languages such as C#, JAVA, VB.net and ASP are commonly used in developing the programming solution. 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, [6] revealed that logical reasoning and problem solving skills can be cultivated while learning to develop a valid workable application.

The preliminary questions asked prior to the software development process are based on the nine topic areas such as (i) business scope of organization, (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. In addition to the information obtained, constraints encountered by students handling software development activities are (i) unclear project scope as users' requirements have been frequently revised, (ii) least user involvement in the designing stage, (iii) lack of management involvement, and (iv) daily processing steps have been ill-defined and often being ignored by staff operating them, and (v) anxiety in handling software projects. 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. In turn, the design and coding activities only take place towards the end of the development schedule plan. 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].

The development of IS project requires an effective participation of stakeholders (users), a comprehensive reviewing of the current tasks flow, understanding of users' requirements and problem encountered. Despite this initial investigation and involvement, software development constraints in relation to external factors and development factors should be emerged. These external factors are still largely controlled by the amount of information obtained from the stakeholders. It has been seen as the responsibility of the developers to obtain the required information. The development factors are generally conceptualized as an application process to incorporate both knowledge and programming skills as well as aptitude in the project development. By reviewing the development factors, this allows the students to assess their strengths, skills and weaknesses. Thus, they will use this information to make subsequent judgment on whether any additional details, further investigation or any refinement in the project proposal is required. The constraint factor model (CFM) presented in Fig. 1 encouraged the students to have a helicopter view of the project aspect and then to zoom into all possible details. With this, students will have the opportunities to construct related programming activities that are based on the analysis findings. From the information gathered, students will actively engage in evaluating their abilities to develop those required skills with an assumption that they intend to undertake this project for success. It becomes the students'

responsibility to manage their learning and take invariably complex and challenging programming tasks.

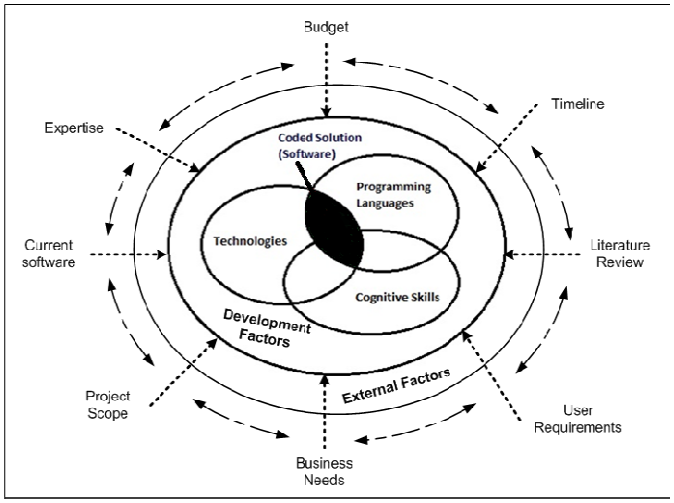

Fig. 1. Constraint Factor Model (CFM) in software development.

By looking at these constraints indicated in the Constraint Factor Model (CFM) 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 analyze 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, and exploratory and experimentory processes of the learning approach.

A. Problem Decomposition in Software Development Stages

Similar to any program solving tasks, developing a program involves steps. The five main stages in the software development process begin with problem analysis, then proceed to planning the solution, coding, testing and end with the documentation. It requires the students to understand the programming process and to incorporate these 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. In this aspect, Fig. 2 presents a problem decomposition software engineering (PDSE) model that synthesizes in-depth evaluation in the development phases.

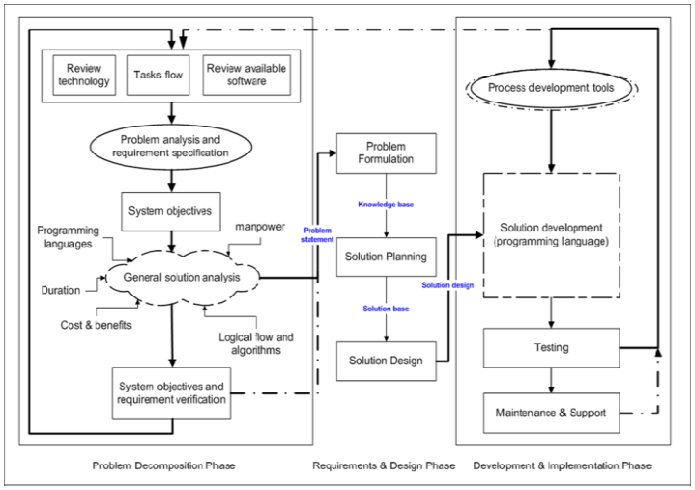

Fig. 2. Problem Decomposition Software Engineering (PDSE) Model in product development process.

The first phase of Fig. 2 is based on the first and second stages in the system development life cycle (SDCL). Software constraints, literature review of processes and technologies are illustrated in the problem decomposition phase. This indicates the tasks that students need to investigate, review and understand before defining requirements. These are the crucial steps of identifying the activities in project development process. Detail analysis through problem decomposition technique allows the students to examine the user requirements more critically by chunking them into manageable section. Engaging with the activity requires the students to have deeper understanding of requirements determined by the users as well as to review their own programming skills and capabilities in constructing and completing the proposed system.

At the second phase (requirements and design), the students formulate solutions based on the gathered information. The problem formulation is derived from the general solution analysis and the comparison of the system objectives against the requirements. It helps the students to analyze the identified problems and user specific techniques, and then to present the problem solution, and thus to validate the solution. This allows the students to determine whether these defined problems and solutions could be implemented successfully or if some form of modification is necessary. At this phase, the students might even review their existing program skills and knowledge which in turn could influence IS development project success. In the development and implementation phase, the students are to apply the most suitable development tools and specific programming languages while converting the solution design into executable solution (system). Testing is carried out to verify for syntactical errors, and then to ensure a fully operational system that the output is in accordance with users' expectation as defined in the initial proposal. Business and user requirements evolve throughout the system usage, which make maintenance and support an ongoing process. As such, the development process begins when changes in expectations are established. In this case, system development is iterative process where new businesses

processes are identified with increase in advance functionalities and complexities of processes. In fact the project success depends on in-depth evaluation at the problem decomposition phase in the PDSE model that derives realistic expectations in order to accomplish the fully operational system within the allocated timeline. Thus, it could somehow assist the students in handling the system development project to transform their performances.

III. Research Questions

In this study, three primary questions have been formulated to address the research outcomes:

1) Is there any significant difference in the number of revisions made to the project proposal between students taught in the DCL and CL instructional methods?
2) Is there any significant difference in self-programming performance between students taught in the DCL and CL instructional methods?
3) Is there any significant difference in software project performance between students taught in the DCL and DCL instructional methods?

IV. Research Methodology

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

A. Research Design

A quasi-experimental design was used to measure the effect of DCL and CL instructional methods on the students' programming development performance. The assessment of students' programming development performance was categorized into three components that include (i) number of revisions to project proposal, (2) self-programming performance, and (3) software project performance. As such, their programming development performance was measured based on the scores obtained from the number of revised project proposal, the self-programming appraisal (SPA) questionnaire, and the software project evaluation (SPE) mark sheet. In this pilot study, a group of 44 students from the second year undergraduate computing course were involved. These two classes, all intact groups, 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). For this study, the course comprised lectures and practical sessions. In classroom, the lecturer reviewed and exploited the various types of system development processes and tools. During practical session, the students applied suitable programming languages in coding.

B. Research Instruments

The students' programming 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 assessment of their development performance includes: (i) number of revisions to project proposal, (2) self-programming performance, and (3) software project performance. The SPA questionnaire was used to measure the students' programming knowledge and skills developed throughout the course semester during the team project activities. 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. Meanwhile, the Motivated Strategies for Learning Questionnaire (adapted from [8]) consists of 23 items was used to identify the students' self-regulated learning level (high or low SRL) prior to the treatment. In this study, the MSLQ mean score of the sample was 3.50. Students who scored 3.50 and above the group mean were categorized as high SRL and those who scored below 3.50 were classified as low SRL. Immediately after the treatment, the SPE was administered to the participants during their software presentation session to measure the students' strategic/conditional knowledge and programming skills. Prior to it, a set of reliability tests were conducted on the instruments used in order to determine the Cronbach's Alpha reliability coefficients. The reliability values of instruments were: (i) 0.90 for the SPE, and (ii) 0.92 for the SPA.

C. The Course Material

Topic imparted to the students during the 28 weeks of treatment (two semesters) was associated with the software project management and development. In the first semester (fourteen weeks), topics related to the basic software project management were discussed with students. In addition to these topics, the problem solving analysis through problem decomposition is covered. For the second semester, students were mainly concentrating on developing the software and the role of lecturer has been switched from lecturing to facilitating and tutoring. This is to ensure that the students achieve all objectives set in the project proposal and complete the project within stipulated timeframe.

D. Data Collection Procedures

The second year semester one computing students in the two intact classes were involved in this 28 weeks study. They were randomly assigned to the two treatment groups (DCL and CL). In this study, the course comprised lectures and practical sessions. To understand software engineering concepts, the students were exposed to the basic software development methodologies (SDM) in the theoretical class session. During the practical session, they were assigned to develop software by using the identified programming languages and the selected software process technique while progressed throughout the development stages.

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 was 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.

V. Research Findings

The SPSS 17.0 for Windows was used in this study. The 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 analyses results are shown in Table II and Table III. The MANOVA results clearly revealed significant differences in all the three dependent variables stated in the research hypotheses.

TABLE II: MANOVA for the Scores of Dependent Variables between the Two Experimental Groups

Dependent variables	df	Mean square	F	Sig.
Number of revised proposal	1	19.88	13.62	0.001*
Self-programming performance	1	2.21	13.10	0.001*
Software development performance	1	686.59	14.68	0.000*

*significant at 0.05 level

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.

TABLE III: Descriptive Statistics of the Dependent Variables for the Two Treatment Groups

Dependent variables	Group	Mean	Mean Diff	SD	N
Number of revised proposal	DCL	1.40	1.35	1.00	20
	CL	2.75		1.36	24
	TOTAL	2.14		1.37	48
Self-programming performance	DCL	3.71	0.45	0.34	20
	CL	3.26		0.46	24
	TOTAL	3.46		0.46	48
Software development performance	DCL	77.60	7.93	6.50	20
	CL	69.67		7.12	24
	TOTAL	73.27		7.85	48

Hypothesis 2: There was no significant difference in self-programming performance between the DCL and CL groups. The result indicated that there was 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.

VI. Discussion

This study aims to investigate the effectiveness of problem decomposition technique used in learning software development process on the students' programming development performance. These second year computing students from two intact groups were randomly assigned to either one of the instructional methods (DCL or CL). One group received the DCL treatment and the other was receiving the CL treatment. During the twenty eight weeks of treatment, the students in both groups worked cooperatively in a team of four members. There were randomly assigned to teams based on their level of SRL, and each team was consisted of two high and two low SRL members. The research findings revealed a significant difference between students taught in the two instructional methods on all the three dependent variables: (i) the number of revisions made to the project proposal, (ii) self-programming performance, and (iii) the project development performance. The analyses indicated that students in the DCL group performed significantly better in making lesser adjustments to the project proposal than their peers in the DL group. These students in DCL group had shown significant result on the capability of self evaluation in terms of their

self-programming performance as compared to those in CL group. Similarly, these students of DCL group significantly outperformed those in the CL group on project development performance. As such, the DCL instructional 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 [9] 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). This technique is seen to be an important step that enables the students to identify and define the software scope before proceeding to development. In accordance with [2], this micro level of identifying and analyzing problems at each IS development process allows the DCL students to visualize software requirements from the users' perspective. Based on user requirements, the evaluation on business processes, realistic expectation and constraints on software project within the project scope spectrum 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. The problem decomposition approach encourages the students to look at possible components and analyse every possibility, thus improving their 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.

Moreover, this problem decomposition technique allows the students to brainstorm on the novel problems that stimulates higher cognitive thinking and in turn cultivates "self troubleshooting" abilities [3][10]. 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. Thus the findings on software development performance have supported previous empirical studies that the adoption of problem decomposition and cooperative learning yielded better quality, fewer defects of codes and shorter development cycles [7][9][11]. This further increases the software project success rate.

VII. Conclusion

The study has emphasized the importance of problem decomposition and cooperative learning technique in learning software development process through handling team project for classroom academic performance. The integration of problem decomposition in information system development cycle has shown significant improvement in completing software projects. 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 during analysis and design stages that eventually to fulfill key aspects towards the software project success. It emphasizes on problem solving through decomposition, system requirements, application development and implementation, which in turn have shown positive influenced in students' programming performance. Likewise, it enables the students to cultivate the essential cognitive skill such as analytical, logical and problem solving skills in the software project environment. Thus, this strategy has 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 when applied in software engineering 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.

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," in *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. Available: 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] R. R. Pintrich and E. V. DeGroot, "Motivational and self-regulated learning components of classroom academic performance," *Journal of Educational Psychology*, 82: 33-40, 1990.

[9] K. Taku and M. Yuji, "Use of support vector learning for chunk identification," In *Proceeding of CONLL-2000 and LLL-2000*, pp. 142-144, 2000.
[10] K. A. Rie and Z.Tong. A high performance semi-supervised learning method for text chunking. 2005. Retrieved July 10, 2011. Available: http://riejohnson.com/rie/ando_zhang_ac205.pdf
[11] C. A. Meseka, R. Nafziger, and J. K. Meseka, "Student attitudes, satisfaction, and learning in a collaborative testing environment," *The Journal of Chiropractic Education*, vol. 24, no. 1, pp. 19-29, 2010.

Tie Hui Hui received her Doctor of Philosophy (PhD) degree in Instructional Technology from the University Science Malaysia (USM), Malaysia in 2011 and holds a Master in IT and Business (MSc in ITBus) degree from the University of Lincolnshire and Humberside, UK. She is a senior faculty member at the Centre for Postgraduate Studies, SEGi College Penang. Her research interests include effective methods of teaching programming, software development and research methodologies.

Irfan N. Umar received his Doctor of Education (Ed.D) degree in Instructional Design and Technology from the University of Pittsburgh, USA in 1999. He is an associate professor at the Centre for Instructional Technology and Multimedia, Universiti Sains Malaysia (CITM USM), Malaysia. His research interest is mainly in instructional design, instructional strategies and e-learning.