SEGi Review ISSN 1985-5672
Vol. 4, No. 1, July 2011, 56-62
*Corresponding author. E-mail: hhtie@segi.edu.my

PROBLEM DECOMPOSITION TOWARDS SOFTWARE DEVELOPMENT: A DATA GATHERING AND ANALYSIS FRAMEWORK

***Tie Hui Hui[1], Irfan Naufal Umar[2]**
[1]SEGi College Penang, Penang, Malaysia; [2]Universiti Sains Malaysia, Penang, Malaysia.

ABSTRACT

Learning programming has always been a challenge for computing students. Programming requires complex cognitive skills such as reasoning, problem-solving and analytical thinking skills that these students seem lacking. College educators who are 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 system development process much more frustrating and difficult. This paper gives an overview of the difficulties encountered and discusses alternative strategies for resolving such affliction, assisting the students towards successful completion and achieving the objectives identified for the system. The problem decomposition and comprehensive data analysis built around the "problem and analysis stages" within the system development cycle will be an effective way of approaching both the process of undertaking programming project and meeting sensible objectives within the speculated timeline.

1.0 Introduction

In learning programming, students are required to understand the given novel scenarios. Once these problem requirements have been identified, programming languages (e.g.: C#, JAVA, VB.net and ASP) are used in developing the programme solution. Problem solving, planning, logical reasoning and analytical thinking are the skills needed in the process of learning and foremost during software development (Miliszewska & Tan, 2007). Dalton and Goodrum (1991) suggested that using the combination of programming and problem solving strategy effectively in course delivery could cultivate problem solving skills. Likewise, Komi-Sirvio and Tihinen (2005) and Maheshwari (1997) revealed that problem solving and critical thinking are both the ultimate skills required in developing a valid workable application. Educators teaching the software project module are constantly seeking for alternatives to ensure that computing students are equipped with the necessary problem solving and programme development skills before venturing into software engineering career. Programming demands complex cognitive skills. As such, an effective course delivery is the key focus in order to develop computing students with high order thinking skills, creativity and innovative ideas. Figure 1 shows a model that blends the problem solving and programme development tasks in the development process. It was created based on the Bloom's (1956) Cognitive Taxonomy, Stenberg's (1985) human information processing and Gagne's (1985) condition of learning. Within the programming environment, crucial knowledge and skills must be nurtured and the expected tasks should be performed at each stage of the phases. This model which comprises of problem solving method, coding tasks and cognitive activities will form the basis for problem solving and programme development in the specified environment.

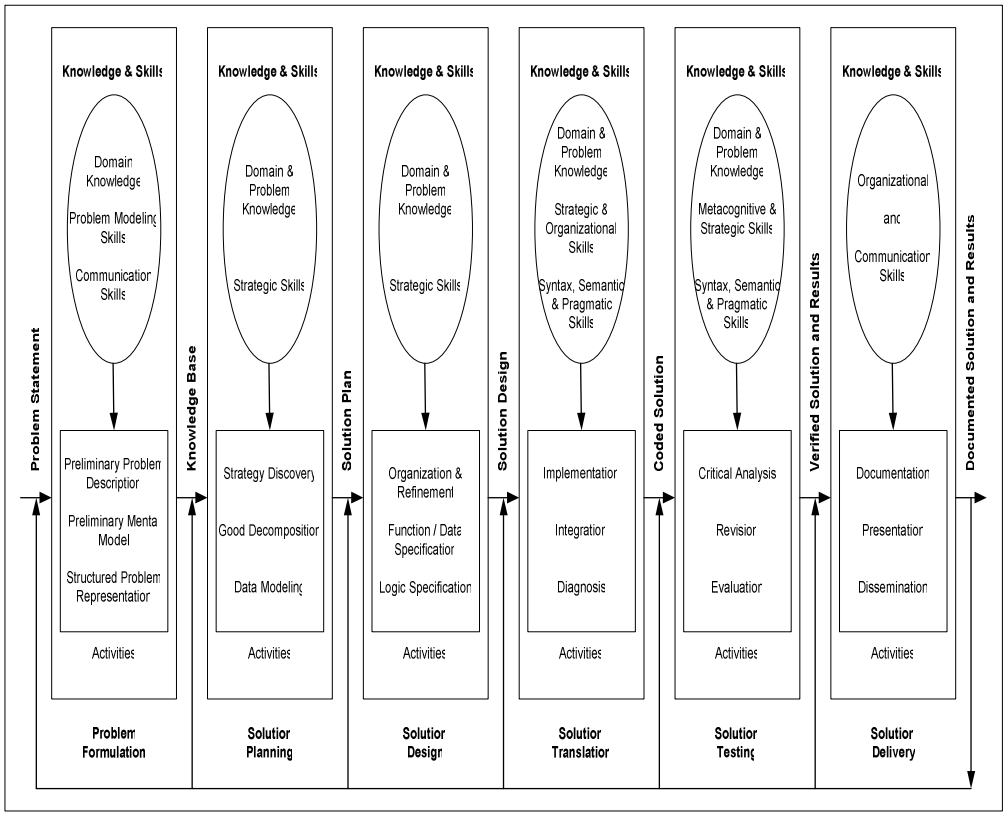

Figure 1: The Cognitive Activities, Knowledge and Skills in the Software-Cognitive Engineering Model (Deek & McHugh, 2003)

Software development activities should be designed to encourage the application of problem solving strategies such as planning, simplification and modelling that incorporates the cognitive system within the development process (Figure 1). In this case, the programming course curriculum should be structured in ways that allow the students to quickly develop a basic understanding of the novel scenario and then move forward to problem solving (that is to conduct the comprehensive findings and to review alternative possible solutions), before producing application. On the other hand, teaching and learning programming in project based environment should be creative, innovative, motivating, challenging and stimulating for both the students and lecturers.

Before software development begins, the knowledge acquisition is to be carried out in the form of a semi-structured questions relating to software development. This set of open and closed questionnaires serves to identify the preliminary problems. The questionnaires cover the four prime topic areas such as (i) business scope of organisation, (ii) business requirements, (iii) user requirements, are sometimes too early for the project sponsors to answer as themselves may be at the preliminary stage of determining the system specification and (iv) resources. While developing the product, students are also encountering challenges that include (i) system requirement, (ii) uncertainty and changes in project scope, (iii) project timeline pressure, (iv) technology compatibility, (v)

functionalities / technical complexity, (vi) anxiety in handling software project and (vii) programming language constraints. As a result, numerous design and coding iteration occur late in the development process, when students find the incompatibility between technology used and the application of programming languages in development stage (solution translation). As expected, the project scope, requirement and constraints (technical and programming tools) are revised throughout the developing cycle. This gives the students insufficient time to implement all identified specifications as stated in the software proposal. Also, they often work under considerable time pressure and may not have sufficient time for critically evaluating the product (Spencer, 2000; Soloway & Spohrer, 1989).

2.0 Problem Statement

Learning to programme is a complex task for computing students. It is not just about learning some programming language syntax. It also involves the handling of software development project. This includes the students' ability to develop algorithm that could solve a given problem scenario effectively. The undergraduate computing students as novice software engineers face difficulties in establishing and adhering to the project objectives within the schedule. Earlier research findings revealed that these students are facing challenges in applying theoretical knowledge and practical skills in the software development (Shahida *et al.*, 2007; Dufner, Kwon & Doty, 1999). These knowledge and practical skills are needed in all phases of system development including analysis, design, coding, testing and maintenance which seem lacking in them. In accordance with Deek and McHugh (2003), lecturers who are currently teaching project module are focusing on language-related activities, with less attention on the earlier tasks of problem identification, system requirements including existing and new resources and details specification. Initial statistical results revealed that 20 percent of the students' project scope had been revised after proposal submission; only few students had completed the system on time and with expected functionality. However, the majority of them were still working on it until the last minutes with incomplete functionality. In turn, limited testing was conducted and some undiscovered errors could have been detected if comprehensive testing strategy was implemented at this stage within the allocated timeframe. These findings are of similar to results reported on IT projects failure in the software industry (Kappelman, McKeeman & Zhang, 2006; Winters, 2002; Jones, 1996).

In addition to problem solving and analytical skills, some root causes encountered by the first author in delivering the course are:

- 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,
- incomplete testing,
- limitation on programming language knowledge,
- incompatibility between technology and programming languages.

These factors somehow increase the risk associated with software development and the probability of failure. For software project to be implemented successfully, a comprehensive understanding and reviewing of the existing user requirements together with the problem decomposition analysis of all possible solutions at each phase (Figure 2) are necessary to be carried out.

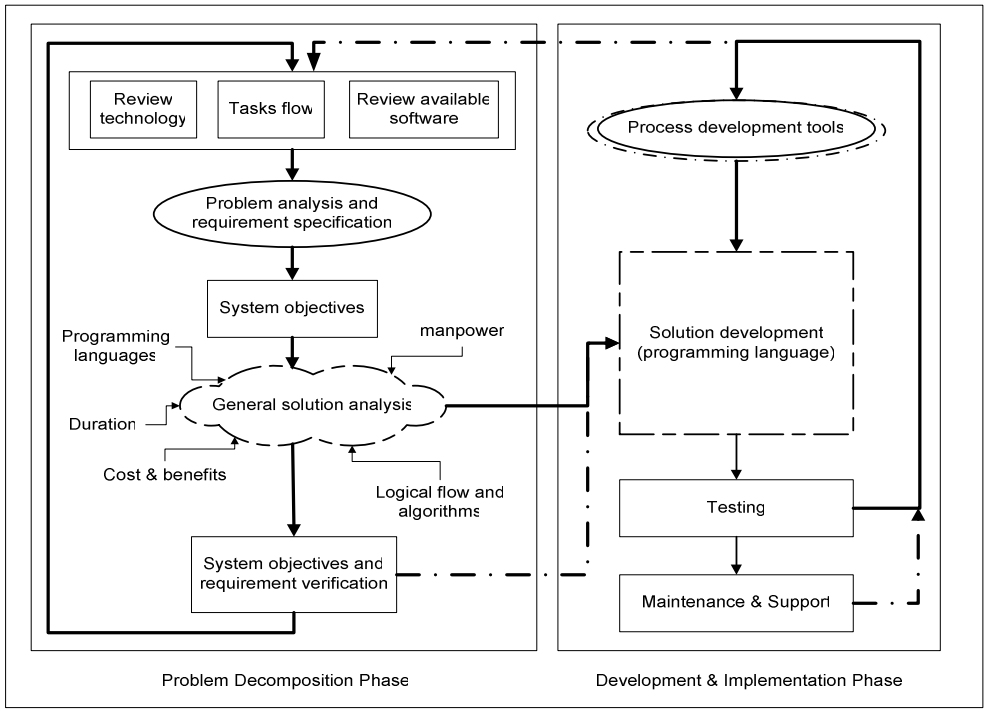

Figure 2: Problems Decomposition in Software Development Process

An Emergent Approach towards Software Developing Process

Developing a programme involves steps similar to any problem solving tasks. Defining the problem, planning the solution, coding, testing and maintenance and following by documentation are the five basic ingredients in software development process. Complex cognitive skills such as reasoning, problem-solving and planning are essential to students in doing the programming projects. Before students undertake a system project, they need to examine and understand the current scenario of the system. Figure 3 depicts the process of collecting and analysing data in more efficient manner that covers from very general to specific and vice versa. This formation of data gathering and analysis framework is derived from the concept of the data-information-knowledge-wisdom pyramid by Fricke (2009).

This framework (Figure 3) is towards a productive requirement gathering, which focuses, prioritises and sequences the set of information needed in understanding the current problems. It allows students to identify the project scope based on the two guided requirement gathering platforms (e.g.: from general to specific) that support problem-solving inquiries. Reviewing the current problems and technical requirement create detail analysis within the requirement and analysis stage of the development cycle. In order to

obtain positive result, the selection and application of various fact finding techniques such as interviewing, observation and record inspection, are to be considered. This data gathering is derived from understanding the current problems and comparing the technical specifications against the desired goals. This comparison analysis will help the students to derive a sensible project scope with practical objectives. In fact, the framework represents a relevant sequence of more effective, productive and realistic ways of acquiring information for critical analysis towards development stages; thus, provide higher success rate in terms of meeting system objectives, business needs as well as completing the product within the stipulated timeline.

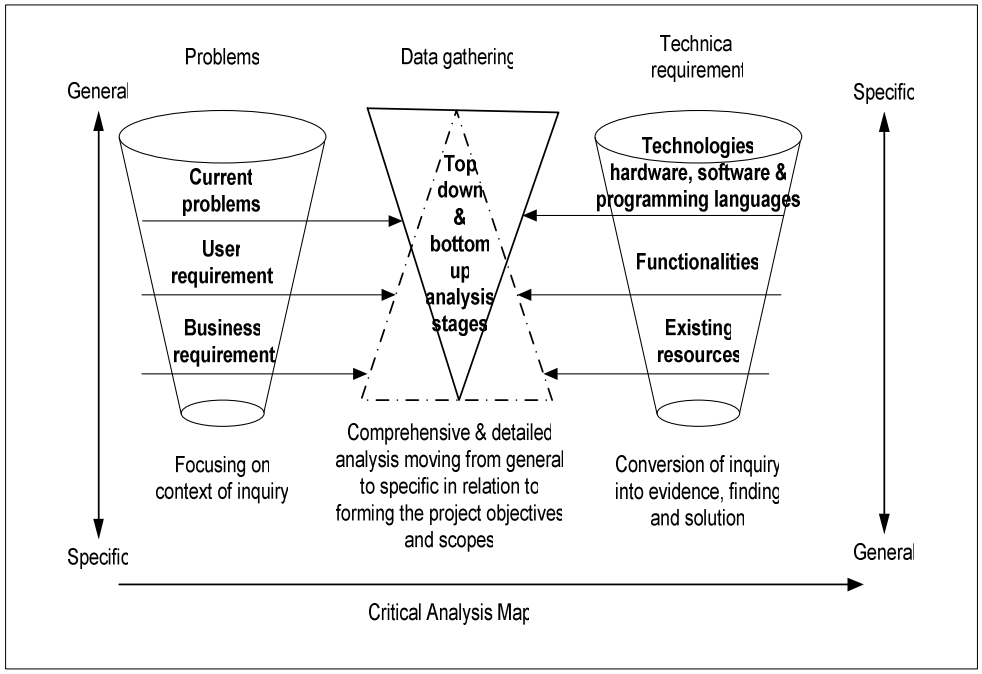

Figure 3: Data Gathering and Analysis Framework

3.0 Summary

Generally, the reasons for software project failure, non-completion or non-fulfilling the objectives set are often related to the range of specific and general requirements. In addition, it also includes the challenges of software, relevant and comprehensive research, and witting documentation. The current method of teaching programming is emphasizing on language-related activities, with less attention on the earlier tasks of problem identification, system requirements (existing and new), resources and details specification. Thus, this paper has outlined a framework and an integration of problems decomposition in development cycle which could be used to reduce failure rate and increase its success. Therefore problem breakdown learning approach during data gathering and analysis stage should be considered as an alternative teaching strategy for lecturers. This approach which focuses on problem solving through decomposition, objective development and application implementation may have the potential to improve 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. As such, both

stages of practical step and general research analysis principles are not only to overcome the software project failure but also for achieving the effective, sustainable and innovative software-building. In teaching and learning programming, this framework could serve as a resourceful guideline for future researchers and educators to promote discovery problem solving skills and to reinforce them throughout the learning activities which in turn will improve overall academic achievement in programming.

REFERENCES

Bloom, B. S. (1956). *Taxonomy of educational objectives, handbook I: Cognitive domain*. New York: McKay.

Dalton, D. W., & Goodrum, D. A. (1991). The effects of computer programming on problem-solving skills and attitudes. *Journal of Educational Computing Research, 7*(4), 483-506.

Deek, F. P. and McHugh, J. A. (2003). *Problem solving and cognitive foundations for program development: An integrated model*. New Jersey Institute of Technology, 266-271. Retrieved April 28, 2011, from http://cblis.utc.sk/cblis-cd-old/2003/2.PartA/Papers/ICT/Deek.pdf

Dufner, D., Kwon, O. and Doty, A. (1999). Improving software development project team performance: A web-based expert support system for project control. *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1-10.

Fricke, M. (2009). The knowledge pyramid: A critique of the DIKW pyramid. *Journal of Information Science*, **35**(2), 131-142.

Gagne, R. M. (1985). *The conditions of learning (4th ed.)*. New York: Holt, Rinehart and Winston.

Jones, C. (1996). *Patterns of software systems failure and success*. Boston, Mass: International Thompson Computer Press.

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

Komi-Sirvio, S. and Tihinen, M. (2005). Lesson learned by participants of distributed software development. *Research Article: Knowledge and Process Management*, **12**(2), 108-122.

Maheshwari, P. (1997). Improving the learning environment in first-year programming: Integrating lectures, tutorials, and laboratories. *Journal of Computers in Mathematics and Science Teaching*, *16*(1), 111-131.

Miliszewska, I. and Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. *Issues in Informing Science and Information Technology*, **4**, 277-289.

Shahida, S., Ahmad, T. K., Zurinahni, Z. and Sarina, S. (2007). System development: What, why, when and how CASE Tools should support novice software engineering. *The 3rd Malaysian Software Engineering Conference*, 256-260.

Soloway, E. & Spohrer, J. C. (1989). *Studying the novice programmer*. Hillsdate, NJ: Lawrence Erlbaum Associates.

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

Sternberg, R. J. (1985). *Beyond IQ: A triarchic theory of human intelligence*. Cambridge, Massachusetts: Cambridge University Press.

Winters, F. (2002). *The Top 10 Reasons Project Fail*. Retrieved May 2, 2011, from http://www.gantthead.com/article.cfm?ID=147229