LEARNING COMPUTER PROGRAMMING IN CDIO'S TEAM SETTINGS

Phuong A PHAM, Man D NGUYEN, Long Q NGUYEN, Thang M NGUYEN, Bao N LE

Duy Tan University, VIETNAM

ABSTRACT

Programming or "coding" skill, for short, is an essential aspect in the training of every Software Engineering or Computer Science program. While certain individuals can quickly digest various programming structures and become proficient in "coding", most others usually go through difficult periods of time trying to understand different programming concepts, grasping certain programming syntax, and applying specific programming algorithms. Eventually, some still manage to "get it" but some others seem to "get to nowhere". So, the question become whether we may teach programming more efficiently, using the CDIO's collective and cooperative learning approaches?

As opposed to what many think that programming can only be learned in an individual manner, many universities are increasingly teaching programming through small team projects in the junior's and senior's level. Assessment of the efficiency and effectiveness of such collective approach, however, are rarely made because of the general assumption that students must have learned programming individually in some introductory programming courses during their freshman's and/or sophomore's year. The whole point of learning programming in teams is sometimes only regarded as a team building activity at a number of universities. At Duy Tan University, during the last two years, we have also attempted to teach programming in teams for freshman and sophomore students. By setting up standard programming projects for teams of 3 to 5 students and by using automated testing tools like PC^2 and Themis, we have created certain flexibility and synergy in the development of programming skills of our students. For teams which did not realize the expected benefits, it mostly had to do with problems in their use of programming language(s) and syntax. At the end of these courses, we carried out an individual comprehensive exam, and to our surprise, the results were significantly better than the introductory programming course, in which programming was taught in an individual manner. Generally speaking, learning programming in teams allow for additional communication and mutual correction on the part of the students and more focused guidance and assessment on the part of our faculty members.

(338 words)

KEYWORDS

Algorithm design, CDIO Standard 4 and 5, learning outcomes, PC^2, programming skills, program testing, Themis

INTRODUCTION

The vision of the CDIO™ Initiative to stress the education of introductory and fundamental engineering coursework in the bigger scope of the whole CDIO (Conceive-Design-Implement-Operate) effort does have its meaningful implications (CDIO™ Initiative, 2010, Standard No. 4). Without good introductory and fundamental engineering coursework, it would be difficult, if not impossible, for educators to deploy an integrated curriculum, integrated learning experiences, and inevitably, the CDIO projects. In the book of "Rethinking Engineering Education", Crawley et al signified the fact that the new approach of introductory engineering courses is to provide students with a complete viewpoint of every little part that makes up their whole four- or five-year program curriculum so that students will understand how to put together those little parts later on as the program progresses as well as to choose the right parts of their interest to become specialized in (Crawley et al, 2007). This approach has been adopted and tested at a number of different institutions, and has proved to be very effective. For example, in their redesign of the Introduction to Information Technology (IT) course at the National University of Vietnam in Ho Chi Minh City, Tran et al provided their students with a complete overview about computer, the Internet, the IT industry, and basic IT skillset besides a systematic hierarchy of IT knowledge (Tran et al, 2013). Or in their systematic re-development of the introductory Mechanical Engineering course, McCartan et al again emphasized the role of the course as "providing fundamental engineering and engineering science concepts" besides developing professional engineering skills for their students (McCartan et al, 2007). Yet, another creative approach was recommended by Lu et al through their introductory engineering course, in which instead of studying the basic theories, students will go through a multi-disciplinary project to learn about engineering (Lu et al, 2010).

While all of the above introductory course design focused on the overall introduction of some engineering discipline as a whole, to every engineering discipline, there is always certain skills that form the backbone. Without a proper level of such skills, students can hardly study or pursue the discipline. Programming or "coding" is one such skill with respect to Computer Science, Software Engineering and any other IT discipline. In fact, at Duy Tan University, around 70% of the students who dropped out of the IT programs attributed their reason to the inability to learn programming or so (DTU, 2013). As a result, at Duy Tan University, we believe that the design of the introductory courses for fundamental skills of an engineering discipline is as important as the design of the introduction course to that whole discipline in the framework of our CDIO deployment. In this paper, we will present how the introductory courses for programming at Duy Tan University were redesigned so as to retain and build the interest of IT students through collaborative learning. Specifically, almost everything about programming would be taught through three different phases, corresponding to three different courses during the freshman and sophomore year, with the use of automated testing tools and in team settings. Initial responses from the students have showed positive results with some interesting findings such as: by learning programming in teams, students pay more attention to the algorithm design and program testing, which is more than often ignored under individual settings; or learning programming in teams of three members produces better results than in teams of four or five; or speed programming competitions in teams help hone students' programming skill in terms of language syntax and coding structure.

CDIO REDESIGN AND DEPLOYMENT OF INTRODUCTORY PROGRAMMING COURSES

In the past, like other IT programs, the introductory programming course at DTU is usually taught at the end of the sophomore year, when students have completed most of the mathematics and natural sciences coursework in their General Education. The traditional reasoning for this approach was that students needed the entire mathematical and fundamental scientific basis before they can move on to programming (McCartan et al, 2007). This reasoning was quite absurd given the fact that Vietnamese students have very good education of natural sciences in high school, and the lag in time during the first couple of semesters in college indeed erodes their interest and knowledge of programming and IT, as a whole, even though they may have learned quite some amount about programming from high school. Hence, the CDIO redesign of our introductory programming courses placed all of them in the first two years with three corresponding subjects:

(1) Phase 1: **Fundamental Programming and Algorithm Basis**, in the course of Fundamentals of Computing 1;
(2) Phase 2: **Data Structures & Algorithms**, in the course of Fundamentals of Computing 2;
(3) Phase 3: **Interactive Programming Exercises**, in the course of Application Development Practices.

The selection of which subjects to be taught in each phase was a complex question because of the many topics to be covered in only three to four semesters. Many faculty members originally complained that we might have gone too fast for the students to digest different programming topics in an orderly manner. As a matter of fact, we had to "test and try" many times before arriving at an arguably optimal combination of certain subjects for each phase. In addition, how to integrate team settings into the learning of programming was difficult because most of the freshman and sophomore students did not have much training in teamwork and interpersonal skills. Yet, we strongly believed in our original reasoning that most of the advanced courses in programming in the junior and senior years would reuse basic concepts of programming and Data Structures & Algorithms so that it is better to teach all those basic concepts to the students at an earlier stage (CDIO™ Initiative, 2010, Standard No. 5). Also, we were confident that teamwork skills can be learned in the well-structured settings of our programming course structures (Nyhoff, 2004). Even though thus far, the teaching of programming in team settings has become quite "painful" and time-consuming to a number of our instructors, the positive feedbacks and results from our students about our redesigned introductory programming coursework was a major encouragement to the entire faculty.

A. Phase 1: Fundamental Programming and Algorithm Basis (in the course of Fundamentals of Computing 1)

In the course of Fundamentals of Computing 1 for the first phase, our goal is to help students grasp and digest various programming and algorithm concepts, and not to focus so much on coding at this point. This is very much different from the standard introductory textbooks about programming, which usually begins with the "Hello-world" class or function, followed by a heavy load of syntax and programming structures of a specific programming language like C++ or Java. The whole idea here is to help students become comfortable with the idea of programming no matter what the language is, and to avoid situations, in which students are obsessed with what programming language they should learn or whether their programs may "run" on the computer or not.

As a result, we allow our students to use pseudo codes, and our grading is based on the general approach of the students' programs rather on the correctness of the programming syntax and structures. To further foster the idea and also to promote our adoption of CDIO, we have developed a nine-step approach for computer programming based on the CDIO Framework, as presented in Table 1. Our students usually call this approach as the "3-3-2-1 approach", (referring to the number of steps in each CDIO phase), and we usually pointed out to them that the actual programming or coding work only makes up one out of nine steps, and if they are not proficient in coding or programming yet, they can still do well in the other steps to ensure the quality of the final product. The "3-3-2-1 approach" actually can be used in any IT courses or projects, which requires programming, not just for the three introductory programming courses mentioned here.

Table 1. "3-3-2-1 Approach" for Computer Programming

CDIO Phase	Problem-Solving Programming Step
CONCEIVE	Step 1: Read and re-read to understanding the Problem. Step 2: Identify the Inputs and Outputs of the Problem. Step 3: Sketch possible Models for solving the Problem.
DESIGN	Step 4: Analyze different details of the Problem. Step 5: Select and consolidate the optimal Model for solving the Problem. Step 6: Decompose the Solution Model into smaller Modules (for coding).
IMPLEMENT	Step 7: Coding or Programming Step 8: User's Testing
OPERATE	Step 9: Test the Reliability & Sustainability of the Program on different systems (with support from the instructors)

Table 2. List of Subjects for Phase 1 - Fundamental Programming and Algorithm Basis

AREA	SUBJECTS
Intrinsic Data Types	Byte, character, integer, real, string, array, memory references (pointers), floating point, etc.
Essential Programming Actions	Program/function/procedure/class/interface/… declarations, value assignment, mathematical operators, functions and subroutines, etc.
Essential Control Structures	Direct sequence, conditional branching, bounded/unbounded iterations and loops, etc.
Subroutines & Recursive Subroutines	Economy of code/solution reuse, abstraction of algorithms, top-down and bottom-up program design
Algorithmic Approaches	Brute force, Divide-and-conquer model, dynamic programming, "greedy" algorithm

Also, right from this first phase, students will be taught about object-oriented programming (OOP) rather than about structured or procedural programming like usual. It has been found that it is very difficult to change the mindset of a procedural programmer into that of an object-oriented programmer (Kölling, 1999) so that it is better to start with OOP first. Moreover, amongst the subjects of teaching as listed in Table 2, we do touch upon areas like Subroutines & Recursive Subroutines and Algorithmic

Approaches. While subjects in these areas like economy of code/solution reuse, abstraction of algorithms, or brute force, divide-and-conquer model, dynamic programming, etc. may sound rather advanced and high-leveled, getting the students familiar with these concepts and structures can go a long way in helping them digest more advanced data structures and algorithms later on (Rosen, 2012).

In terms of team learning, in this phase, we mostly give out group homework to allow for collaboration amongst the students. To facilitate strong and clear communication between the students, we set up a Facebook site where students can privately discuss different issues with their team members as well as publicly with members of other teams. Compared to the traditional online forum format that we usually used, Facebook offers many more features and utilities for private and team communication. We have found many situations, in which students are too shy to ask a question or to share their idea in class, but they have become very active on Facebook, contributing in many technical and non-technical discussions about programming.

B. Phase 2: Data Structures & Algorithms (in the course of Fundamentals of Computing 2)

In the second phase of Data Structures and Algorithms, students are introduced to more advanced subjects (Table 3), but largely, they still work on the programming problems and subjects of the first phase. The main goal of the second phase is to hone the students' coding skills since they have become familiar with different programming concepts after the first phase. This means we will come right down to the syntax and other details in the codes of our students, and another major goal for our students is to develop their debugging skills. The optimal approach for this phase as we found out was the adoption of speed programming competitions amongst the teams of three to four students. The clear edge of speed programming competitions over group homework or big team project is that the students have to be very precise in every little detail and be punctual in their work. This is not to mention of another requirement that the students have to effectively distribute the work amongst their team members, for example, one person will formulate the program structures while another will follow suit and do the typing, yet another will think of the solution for the next problem, so on and so forth. The format very much resembles the ACM/ICPC competitions (ACM/ICPC, 2013), but in order to relieve the amount of pressure on the students participating in the competitions, we assign very small grade percentiles on their performance for these competitions. However, we make many of these competitions throughout the semester with the idea of helping the students overcome their fear of competition and time constraint to build up their own programming skills.

Table 3. List of Subjects for Phase 2 - Data Structures & Algorithms

AREA	SUBJECTS
Advanced Calculus	Number theory, counting methods, Pigeonhole principle
Graph Theory	Graph presentations, shortest-path algorithms, minimum spanning trees
Data Structures & Analysis	Array, vector, list shorting, linked list, stacks and queues, hash and hash tree

The team settings of the second phase has led to the need to move our class sessions to the computer labs (CDIO™ Initiative, 2010, Standard No. 6), and the adoption of Themis, an automated scoring software. Instructors also need certain LAN and classroom monitoring tools like NetOp School or Vision. Students' requests for online Q&A after class with their instructors and teaching assistants were also reported.

C. Phase 3: Interactive Programming Exercises (in the course of Application Development Practices)

While the speed programming competitions used in the second phase were effective in honing the students' programming and debugging skills, they did not provide students with in-depth knowledge about real-world programming work because most of the problems in these competitions are small ones, which can be completed in a matter of minutes or hours. As a result, in the third phase, we deploy bigger programming problems, which are due on a weekly basis, and students will also focus on these weekly assignments without having to study any new subjects.

The move to weekly online programming assignments, however, posed certain technical problems. We had to drop Themis and adopt PC^2 for the course of Application Development Practices. PC^2 is also an automated scoring software like Themis, but it is much more complicated and can link up different networks for its automated scoring functionality. PC^2 modules on different clients and servers communicate with each other over TCP. So, if the clients and servers are all on the same network, all the connections should work as long as the servers use publicly routable IP addresses. If, however, different sets of clients and servers are on different networks, certain problems will arise out of network configurations of firewalls, VPNs (Virtual Private Networks), NAT (Network Address Translation), etc. For example, for PC^2 to work with NAT, we need to configure a forwarding port on the firewall of our network, accompanied by a site table with public addresses and ports for remote connections from outside in. As a result of all these problems, we generally encourage our students to upload their work to PC^2 from the school's network or at least, VPN in the school's network to upload their work.

For the team settings in this third or final phase of introductory programming, again most teams have three to four students with some exceptions of up to five students. Surprisingly, there was not much difference in the performance between teams because of the number of team members. The difference in performance usually came from the number of excellent students that certain teams possess. In addition, even though we did not ask the teams to assign their team leader and other roles, some teams independently assigned roles among their members for better performance. It is interesting to see how teamwork skills are independently learned by the students under certain circumstances. As for the technical issues of network connections, they were not as bad as we predicted, and the increasing number of our students who have laptops, however, made it easy for us to administer this final phase of introductory programming.

RESEARCH RESULTS & DISCUSSIONS

Our redesigned introductory programming courses have been deployed during the last three years within the scope of the Software Engineering program of the International School at Duy Tan University. The Faculty of Information Technology of DTU, on the other hand, still follows the traditional approach with another set of three similar programming courses of

Basics of Programming (sophomore year), Object-Oriented Programming (junior year), and Data Structures & Algorithms (junior year). While the Basics of Programming course is an introductory one, Object-Oriented Programming and Data Structures & Algorithms are considered more of an advanced level. Still, for the purpose of assessing the effectiveness of our redesigned introductory programming courses, we selected a random group of 50 students, who have taken the three traditional programming courses, and another random set of 50 students who have taken the three redesigned introductory programming courses. These students are given an individual comprehensive examination about programming to test their programming skill and capability. The students will be rated by a letter grade according to the grade percentile breakdown listed in Table 4.

Table 4. Letter Grade and Percentile Breakdown

Letter Grade Rating	Grade Percentile Breakdown
A	85% - 100%
B	70% - 84%
C	55% - 69%
D	40% - 54%
F	< 40%

It should be noted that the percentile grade brackets for A, B, C, etc. letter grades in Vietnam are considerably lower than those of Western countries. There is a long history about this fact. Basically, before the higher education system in Vietnam adopted the credit-hour system, we followed a class-based cohort system, and we did not assign letter grades but only numerical grades out of the scale of 0 to 10. The teachers and instructors in Vietnam in the past rarely gave out the maximum grade of 10 or even 9 with the reason that a student must have been as good as the teacher to deserve such a grade. As a result, the actual maximum grade was usually 8. Ever since we adopted the Western-styled credit-hour system, we have tried hard to move away from this norm; yet, many instructors are familiar with it. Hence, the Ministry of Education & Training (MoET) has issued a number of guidelines about grade letters and the corresponding percentile brackets - Table 4 above is, in fact, the formal guideline provided by the MoET.

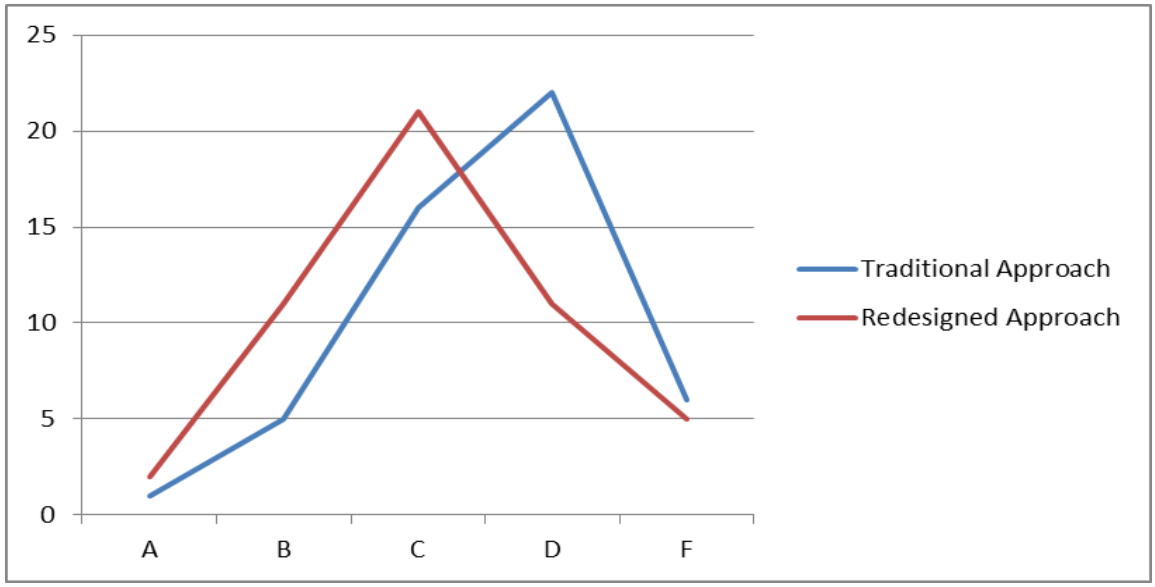
Table 5. Grade Distributions for Traditional Approach and Redesigned Approach
(by the number of students)

	Letter Grades				
Approach	**A**	**B**	**C**	**D**	**F**
Traditional Introductory Programming Coursework	1	5	16	22	6
Redesigned Introductory Programming Coursework	2	11	21	11	5

The results from our individual comprehensive examination were very positive with regards to our redesigned introductory programming courses. While there is not much difference in the number of students getting A (or F), there were a significant difference in the number of students who get B (11 compared to 5 - an increase of 120%) and a sizeable drop in the number of students who get D (11 compared to 22 - a decrease of 50%). Students who took the redesigned introductory programming courses also received more C than those who followed the traditional track, by a difference of 5 students. Considering the fact that the

students with the redesigned approach are those who have just completed their sophomore years and students with the traditional approach are the ones who have finished their junior year, this is indeed a significant improvement in terms of the students' programming skill and capability. As Figure 1 demonstrates, there is a clear shift to the left for the grade distribution of students who took the redesigned introductory programming courses.

Figure 1. Grade Distributions for Traditional Approach and Redesigned Approach



To further assess the strengths and weaknesses of our redesigned introductory programming coursework, we carried out another survey with the 50 students who have taken those courses. The goal was to assess the relevance of our materials, the effectiveness of our instructors, the performance of the automated scoring software, the improvement in students' programming skills, so on and so forth within the scope of the three courses of Fundamentals of Computing 1, Fundamentals of Computing 2, and Application Development Practices. Table 6 shows the 14 statements we used in our five-point Likert-scale questionnaire for the survey.

Figure 2. Students at DTU Learning Programming in Teams

Table 6. Questionnaire for Assessment of the Redesign of Introductory Programming Coursework

Duy Tan University						
Student ID:	Student 's Name:	Date:				
		Tick One				
5: Strongly Agree - 4: Agree - 3: Neutral - 2: Disagree - 1: Strongly Disagree						
No.	Survey Statement	5	4	3	2	1
1	Studying computer programming in teams helps foster algorithm analysis, creativity, and mutual learning.					
2	The three introductory programming courses (Fundamentals of Computing 1 & 2, Application Development Practices) help develop your skills in finding the optimal solution for programming problems.					
3	The course of Fundamentals to Computing 2 helps develop your skills in finding quick solutions for programming problems.					
4	Your overall programming skills have improved as a result of the three introductory programming courses (Fundamentals of Computing 1 & 2, Application Development Practices).					
5	The instructor in the course of Fundamentals of Computing 1 was always supportive of the students.					
6	The instructor in the course of Fundamentals of Computing 2 was always supportive of the students.					
7	The instructor in the course of Application Development Practices was always supportive of the students.					
8	The instructor in the course of Fundamentals of Computing 1 always encouraged creative thinking and active learning.					
9	The instructor in the course of Fundamentals of Computing 2 always encouraged creative thinking and active learning.					
10	The instructor in the course of Application Development Practices always encouraged creative thinking and active learning.					
11	The automated scoring software used in the course of Application Development Practices was working properly.					
12	The group homework and team assignments in the three introductory programming courses (Fundamentals of Computing 1 & 2, Application Development Practices) were practical to your future career.					
13	The group homework and team assignments in the three introductory programming courses (Fundamentals of Computing 1 & 2, Application Development Practices) were graded fairly.					
14	You will recommend the three introductory programming courses (Fundamentals of Computing 1 & 2, Application Development Practices) to other students.					

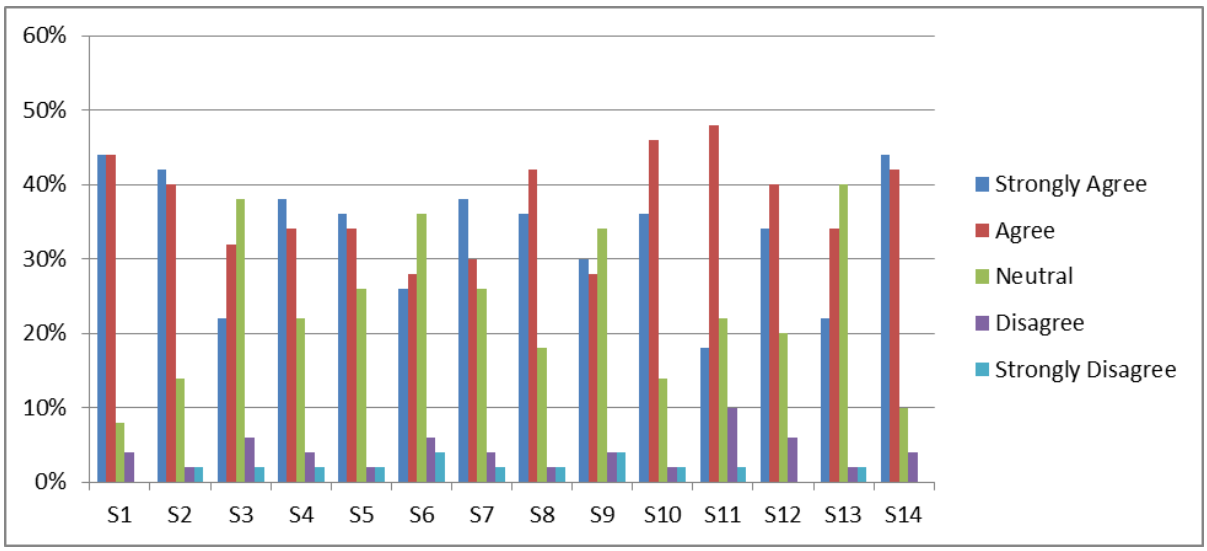
The results of the survey (Table 7) were very positive with strong levels of agreement to most statements. However, there are also certain insights that we need to pay additional attention to. The highest levels of "Strongly Agree" and "Agree" went to Statements 1, 2, 10 and 14, signifying the fact that the three introductory programming courses did help improve the programming skills and creative analysis of our students. On the other hand, amongst those three courses, Fundamentals to Computing 2 appeared to receive the lowest levels of interest from the students with comparatively low total percentages of "Strongly Agree" and "Agree" to Statement 3, 6 and 9 even though these totals were still considerably bigger than the levels of "Disagree" or "Strongly Disagree" to those statements. It seemed that the high-

level concepts of Data Structures & Algorithms together with the speed programming competitions did put quite some uneasy pressure on the students. In addition, we definitely need to improve on the quality of teaching of our instructors for this course. There also seemed to be quite a number of technical problems with the automated scoring software since up to 12% of the students disagree or strongly disagree to Statement 11. In addition, the most number of "Neutral" responses (40%) about fair grading went to Statement 13, suggesting that we may need to be easier on grading to create more encouragement.

Table 7. Students' Feedbacks about the Redesigned Introductory Programming Coursework

Statement No.	Strongly Agree		Agree		Neutral		Disagree		Strongly Disagree	
1	22	44%	22	44%	4	8%	2	4%	0	0%
2	21	42%	20	40%	7	14%	1	2%	1	2%
3	11	22%	16	32%	19	38%	3	6%	1	2%
4	19	38%	17	34%	11	22%	2	4%	1	2%
5	18	36%	17	34%	13	26%	1	2%	1	2%
6	13	26%	14	28%	18	36%	3	6%	2	4%
7	19	38%	15	30%	13	26%	2	4%	1	2%
8	18	36%	21	42%	9	18%	1	2%	1	2%
9	15	30%	14	28%	17	34%	2	4%	2	4%
10	18	36%	23	46%	7	14%	1	2%	1	2%
11	9	18%	24	48%	11	22%	5	10%	1	2%
12	17	34%	20	40%	10	20%	3	6%	0	0%
13	11	22%	17	34%	20	40%	1	2%	1	2%
14	22	44%	21	42%	5	10%	2	4%	0	0%

Figure 3. Students' Feedbacks about the Redesigned Introductory Programming Approach



With respect to the learning outcomes for the students, feedbacks to Statements 1 to 4 and 11 to 14 provided a general positive picture with the exception of certain shortcomings still identified with the course of Fundamentals to Computing 2. To gain a deeper understanding of the actual learning outcomes achieved as a result of our team-based learning of programming skills, we further held some qualitative discussions with the students, and these were some of our findings: (1) Students commented that they now do not have to rely

so much on the guidance of the instructors, but rather, can automatically and voluntarily learn from their programming partners and other online sources. (2) Students also appeared to spend more time to learn about programming tricks and hacks to optimize their codes, which is different from past experiences when they only spent time trying to understand complex programming structures. (3) A sizeable number of (below-average) students still did not get a good grasp of complex programming structures, but interestingly, they can still implement those structures in their codes by following and imitating what others are doing. (4) Most students stated that their ability in asking the right questions for programming problems at hand has been greatly enhanced through the use of Design and Implement brainstorming schemes. (5) A sense of "everything is doable in programming" has become prevalent amongst the students as they said they can now get help anytime from their programming partners, friends, instructors, or anyone with certain amounts of programming knowledge.

CONCLUSIONS

While there are still many things that we need to work on to improve the redesign of our introductory programming courses, the good news is that our redesign did, in fact, bring values to the study of computer programming of our students and significantly shorten the amount of time it takes for them to learn programming. By the end of the sophomore year, most of the students in the Software Engineering program of our International School have become quite proficient in programming whereas that can only be achieved by the end of the junior year for students in the Faculty of Information Technology. For our instructors, the general feedback we got was that they had to spend amounts of extra time and effort with the redesigned courses. This is understandable given that it is not easy to teach programming in team settings. Additional compensation and better workspace/technology should be down the road to make their work easier.

REFERENCES

ACM/ICPC (2014). *ACM/ICPC World Finals*. http://icpc.baylor.edu/worldfinals

CDIO™ Initiative (2010). CDIO Standards v. 2.0. http://www.cdio.org.

Crawley, E.F., Malmqvist , J., Ostlund, S., and Brodeur, D. (2007). *Rethinking Engineering Education: The CDIO approach"* (1st Ed.). Springer.

Duy Tan University - DTU (2013). Students' Placement Report. Vietnam: August 2013.

Kölling, M. (1999). *The Problem of Teaching Object-Oriented Programming, Part 1: Languages*. Journal of Object-Oriented Programming, 11(8), pp. 8-15.

Lu, X., Fan, Y., Banzaert, S., Jacobs J. (2010). *Multi-Disciplinary Design-Build PBL As An Introduction to Engineering*. École Polytechnique, Montréal, Canada: Proceedings of the 6th International CDIO Conference: June 15-18, 2010.

Nyhoff, Larry (2004). *ADTs, Data Structures, and Problem Solving with C++, 2nd Ed*. U.S.: Prentice Hall, 2004.

McCartan, Charles D., Cunningham, G., Bernard, E., Buchanan, Fraser J., McAfee, M., Kenny, Robert G., Taylor, I., and Mannis, A. (2007). *The Systematic Development of the New Introductory*

Course. Massachusetts Institute of Technology, MA, U.S.: Proceedings of the 3rd International CDIO Conference: June 2007.

Rosen, Kenneth H. (2012). *Discrete Mathematics and its Applications, 7th Edition*. U.S.: McGraw-Hill Science/Engineering/Math, 2012.

Tran, Son T., Le, Thanh N., Nguyen, Binh Q., Dang, Phuong B., and Le, Bac H. (2013). *Introduction to Information Technology*. Massachusetts Institute of Technology and Harvard University, MA, U.S.: Proceedings of the 9th International CDIO Conference: June 2013.

BIOGRAPHICAL INFORMATION

Phuong A Pham is a lecturer of the Faculty of Information Technology of Duy Tan Univercity. He is in charge of the Information System Academic Group. His interests are in Computer Graphics, Pattern Recognition, and Image Processing.

Man D Nguyen is the Acting Dean of the International School of Duy Tan University. His majors are in Software Engineering and Information Systems Management. He has more than 10 years of experience in software development and mentoring for Capstone projects. His interests are in software testing, mobile application development, and large-scale data processing.

Long Q Nguyen is a lecturer of the Faculty of Information Technology of Duy Tan University. His interests are in Analysis and designed systems, ERP and data warehousing.

Thang M Nguyen is an instructor in Information Systems of the International School, Duy Tan University. His interests are in software project management, software process, business decision-making, and IT in business.

Nguyen-Bao Le is the Vice Provost of Duy Tan University. He is in charge of the Technology & Science division as well as the R&D Center of DTU. His interests are in data warehousing, OLTP, graphics and animation design.

Corresponding Author

Mr. Phuong A Pham
Head of Academic Group,
Faculty of Information Technology,
Duy Tan University
K7/27 Quang Trung, Da Nang
Vietnam 59000
+84 914019099
phamanhphuong@dtu.edu.vn