Journal of Contemporary Issues and Thought
ISSN 2232-0032/ e-ISSN 0128-0481/ Vol 12, Issue 1, 2022 (65-79)
DOI: https://doi.org/10.37134/jcit.vol12.1.6.2022

The Effectiveness of Project Based Learning with Computational Thinking Techniques in a Software Engineering Project Course

Aslina Saad

Department of Computing, Faculty of Art, Computing and Creative Industry,
Universiti Pendidikan Sultan Idris, 35900 Tg. Malim, Perak, Malaysia.
Email: aslina@fskik.upsi.edu.my

Abstract

To date, finding the most effective method to teach SE students is extremely challenging. This study used the project-based learning approach which applies computational thinking techniques by engaging students via inquiry based, hands on and student centric in teaching the software engineering Project course to students enrolled in a software engineering program. The objective of this study was to evaluate the effectiveness of the project-based learning approach as an innovative pedagogical method to improve software engineering students' skills in three critical domains, namely practical, knowledge, and soft-skills. Seventy students' skills and knowledge of various aspects of software engineering before and after the learning intervention were compared using the t-test. They were guided to carry out a software engineering project in several groups. Survey questionnaires consisting of close and open-ended questions to probe various aspects of software engineering, namely project planning, requirement analysis, software design, developments, and presentation, were administered to the students after the learning intervention. The results showed the differences in students' skills and knowledge in the above aspects of software engineering were significant, signifying that the project-based learning—computational thinking approach is an effective approach to help students learn such aspects of software engineering more efficaciously and boost cooperative learning.

Keywords: Project-based learning; Computational thinking; Software engineering education; Software project

1. Introduction

Over recent years, the enrollments in software engineering (SE) programs in universities all over the world have been rapidly increasing, as job prospects associated with software development are currently in high demand. Notably, in the last few years that witnessed the rapid growth of IR 4.0, these programs have become one of the popular academic programs offered by many institutions of higher learning, which are primarily attributed to pervasive use of software that has radically altered various spheres of people's lives, such as education, management, business, and many more. As such, the quality of peoples' lives has significantly improved by the use of a vast range of software applications. In this regard, the quality of software application needs to be assured to meet stringent user requirements, which is a very challenging feat to achieve given the continually changing character of software production.

According to Santos (2009), SE professionals face many new challenges, entailing them to adapt new concepts, methods, and techniques to deal with specific situations of the SE market. In general, the challenges are influenced by the variations of scope, cost, development time, and risks of software projects. Therefore, as suggested by Shaw (2000), a new educational perspective is needed to help train and prepare students with different

roles by infusing a stronger engineering attitude in the current curriculum, which can help students stay current in the face of rapidly changing landscape and establish strong credentials that accurately reflect their abilities. Furthermore, Sindre (2018) asserts that the preparation of IT graduates for professional careers centers on real-life experiences.

Admittedly, a software engineer needs holistic skills, entailing comprehensive training beyond core software engineering development (such as requirement elicitation, analysis, design and development) by emphasizing project management, team building, progress tracking, change management, and communication. Such training is in line with Shaw's (1999) assertion that in the real world of software development, software engineers have to deal with multiple tasks and job scopes, such as project manager, requirement engineer, system analyst, software designer, programmer, software tester, and quality engineer.

The various skill of a software engineer, from project management, requirement modeling, software design, programming and software testing were introduced to the students during their four (4) years enrollment in SE program. These various skills could be instilled in the student by using various Computational Thinking (CT) techniques as CT is popular for its problem-solving capability. Yadav (2014) suggests that CT has the potential to advance students' problem-solving skills and abilities significantly, with which they could think in new ways. Saad (2020) stated that (CT) is one of the important skills that all individuals must have for problem-solving and decision-making through a systematic approach. These techniques can be used to complex problems more accurately and these ultimately enable the students to formulate quality solutions efficiently.

In principle, SE is a practice-oriented, applied discipline that emphasizes on hands-on development, process, and use of tools in addition to theoretical concepts. Finding from Saad (2020) shown that three learning approaches, namely student-centric, inquiry-based, and hands-on learning approaches are effective in developing strong CT skill among young students (primary school pupil). This possibly has the same impact for students at tertiary education level.

According to Mittal and Sureka (2014), balancing theoretical and practical aspects is a tall order in Software Engineering (SE) education. Similarly, Souza et al. (2019) affirm that like any other engineer, a software engineer must master the theoretical foundations, design methods, and the technology and tools of the discipline. Furthermore, they argue that current teaching practices lack emphasis on the development of soft skills, which is partly attributed to the difficulty in training students to acquire such skills. In addition, Frovanti et al. (2018) contend that the teaching of theoretical concepts without practical applications or examples in proper contexts may undermine learning efficacy. Cooperative learning can be fostered via student-centric and hands-on approach while, inquiry-based approach may cover the theoretical aspects.

Gillis and Boyle (2011) in their study revealed that, cooperative learning is a pedagogical practice that has attracted much attention over the last three decades, as a large body of research has indicated that students would be able to gain several benefits, both academically and socially, when they are given the opportunities to interact with one another to accomplish a number of shared goals. Surely, students must be able to keep their knowledge current with respect to new approaches and technologies, interact with other people, and model, formalize, and analyze a new problem. Moreover, they need to recognize recurring problems and reuse or adapt known solutions, manage processes, and coordinate works with different people. Such a notion is echoed by Charlie et al., (2009), who claim that SE education must focus not only on technical skills of software

development but also on communication and interaction among learners. As such, the conditions of practical training in relevant principles, methods, and procedures must be similar to those of software development, if SE education is to be efficacious (Yadav and Xiahou, 2010). The group-based project for this MES3073 course could foster cooperative learning in order to finish their project and managed to improve communication skills among the team members.

Having reviewed The Engineer of 2020, Hazzan and Dubinsky (2007) underlined the importance of a well-rounded skill set, some of which are better suited to be taught and learned at school while others can be acquired through real-world practical experiences. In this regard, they claim that SE differs from the other engineering fields in that the theoretical foundations of the former are less mature, often far from being directly applicable, and hence less immediately useful.

According to Ghezzi and Mandrilo (2005), the technical skills that a software engineer should possess can be developed by integrating course teaching with projects, which has become a contentious issue in the SE education, with many teaching practitioners lamenting that replicating the complexity of real-life projects in an educational environment can be virtually impossible. As such, project-based learning can offer ways to transfer the learning of foundational and practical knowledge to real projects involving real clients. However, there is no consensus on how to efficaciously teach SE courses, as lecturers who come from many different institutions adopt their own teaching methods and approaches based on their experiences.

Effectively, project-based learning (PBL) is a comprehensive method that focuses on teaching by engaging students in investigations (Blumefled, 1991). This approach involves students in designing, developing, and constructing hands-on solutions to a given problem. Through such real-world problem activities, students are guided to solve problems systematically using appropriate skills, tools, and techniques. However, Gary (2018) cautions that a project process is different from a learning process that most students find challenging. In this respect, Souza (2019) suggests that student-centered approaches may be more suited for the development of students' competence as the 'learning-by-doing' approaches help motivate students to learn as they play an active role in the learning process. PBL truly matches the student-centric, inquiry based and hands-on mechanism of CT.

Lack of research has prompted the researcher to conduct a study to examine such an impact. In this study, the SE Project course was carried out by integrating PBL with CT technique. Such an integrated PBL approach provided a common framework to guide students to learn appropriate concepts, skills, tools, and techniques in a proper context. The combination of the CT techniques within PBL was important to balance both theoretical and practical skills encompassing three learning domains, namely psychomotor, cognitive, and affective domains. The four main techniques of CT; decomposition, algorithm, abstraction and pattern recognition were applied in various aspects of software engineering project.

Although PBL and CT has become a very hot topic in educational research and practice, specific work relating to the combination of this approach is still in its infancy. In response to the lack of such research, several researchers have begun to dedicate their studies to computational thinking involving students, however the integration of PBL and CT in education is still limited. The aim of the study was to define guidelines for teachers to help them design a series of lessons aimed at developing software engineering skills and knowledge with the use CT and PBL. Furthermore, the evaluation of its implementation

investigated the effectiveness of PBL-CT approach in improving students' skills and knowledge in SE area.

2. Research Methodology

Participants

Seventy (70) students enrolled in the SE Project course in the first semester of the 2019-2020 academic session were recruited for this study. They were equally divided into three groups who learned the course for 14 weeks, with each week consisting of PBL-based learning sessions that lasted for three hours. Three CT learning approaches, namely student-centric, inquiry-based, and hands-on learning approaches were introduced in this class using PBL teaching method.

Instrument

Survey questionnaires were distributed to all students at the end of the semester to help evaluate the effectiveness of this novel learning approach. Specifically, they were required to take an online survey consisting of questions pertaining to various aspects of the PBL-CT implementation, and their knowledge and skills that they acquired in various phases that they had gone through in finishing their projects. The survey consisted of three section namely the demographic, knowledge and skills and input/suggestion.

Statistical test

The instrument was analysed using descriptive statistical tools and inferential statistics. Pre and post-test test were used in this study to examine the change in various aspects of software and its engineering with the PBL-CT approach. The pretest was done in the beginning of semester and a posttest toward the end of the semester. Student learning can be inferred from the, difference in student performance between two points in time, at the beginning of the semester and at the end of the semester. The scores were analyzed as a whole (group) rather than looks at changes within the individual.

The paired samples t-test provides an estimate of the significance of the difference between the means of the two samples from the same subjects – in this case, the pre- and post-tests of the students in a class. The p-value that the test provides gives us information that allows us to accept or reject the null hypothesis. If the test is significant at $p < .05$, we can reject the null hypothesis and ascertain that the pre- and post-test differences are different, and the differences is not caused by chance.

Hypothesis

The null hypothesis is there was no significant difference in the level of SE skills and knowledge before and after the software engineering project class.

The PBL-CT Implementation

Out of the three-hour sessions per week, one hour was dedicated to help the students to learn relevant theories, principles, tools, and software to carry out a number of tasks and activities. The remaining two hours were allocated to help the students to carry out several

PBL activities in order to accomplish a given project with the lecturer's guidance, entailing the learning activities to be in line with the learning outcomes.

The three learning outcomes (LOs) of this course and the learning process that they were required to follow were explained to the students in the first learning session. The first LO entailed students to produce complete software documents for software development which primarily involved the cognitive domain. The second LO required the students to develop specific software based on the produced documents, which aimed to help them acquire appropriate practical skills. The third LO compelled the students to demonstrate professional ethics in software development, which involved the affective domain, namely soft skills comprising communication, leadership, and interpersonal skills and others.

In the first learning session, the students were asked to identify a suitable real-world project in the educational domain. In particular, this project should relate to education, as the specialization of this SE program is educational software. The project should be completed within 14 weeks (one semester). They were required to brainstorm several ideas of the software project to be developed in several groups, each consisting of three (3) members. According to Bell (2010), the implementation PBL can help prepare students to meet the challenges of the twenty-first century, equipping them with a repertoire of skills that can help students learn successfully. More specifically, he argues that such skills can make students become productive members of a software development project.

In the 1st class, the students were asked to share some ideas that could be used in this project by focusing on the current trend in the SE field, which could be obtained from governmental initiative, conferences, fairs, exhibitions, research papers and training provided by various organizations. They were required to present their projects ideas that would be reviewed by their peers. In principle, the students were required to write a simple proposal containing the justification for the development of software. The abstraction concept and logical reasoning of CT are applied in preparing the proposal as the students are required to highlight the problem statement, objectives and scope of the project. Abstraction in CT enables us to navigate complexity and find relevance and clarity at scale. An abstraction figure out how to work with the different parts efficiently and accurately and this is important in determining the statement of problems, objectives to be achieved and scope of the project. Their course mates were welcomed to ask, give inputs and comments to help them improve their proposals and this foster cooperative learning not only among team members but also with their classmates. In addition, they need to classify their proposed project to categories of software; web-based system, mobile application, Augmented Reality, Virtual Reality, IoT, data mining or robotic. In this presentation, the lecturer triggered some questions as inquiry- based approach in CT to introduce the concept of pattern recognition which is about observing patterns, trends, and regularities among and within problems, in order to make classification of the proposed software's category.

After making such an improvement to the proposal, they had to choose an appropriate Software Process Model (SPM), a variation of the software development life cycle (SDLC), which is crucial as a guideline to implement their project. The selection of the SPM should also be justified based on various factors, such as budget, time, clarity of the requirement, scope of the project and familiarity of the technology that will be used. In order to explain the concept of SPM, the students were shown a video related to Smart Home System. Based on the video, each of them needs to write something which relates with software project on the Mentimeter, an interactive presentation application. Here, the abstraction technique of CT was instilled whereby they need to elicit relevant concepts, theories, models, methods or

techniques in SE and the outcome of the Mentimeter was then shared with all the students. They were also asked whether all the terms can be classified into several main groups in SE. Based on their answers, the lecturer showed a table as illustrated in Table 1, consisting of several main groups namely phases, activities, models, tools, techniques and deliverables. The concept of decomposition in CT was introduced here by breaking down the software process model into smaller, manageable parts.

Table 1: The decomposition of a software process model

Phases	Activities	Model/ tools/ techniques	Deliverables
Phase 1	• Activity 1a • Activity 1b	• Any model, tools or techniques of an activity need to be listed. • Model, tools or techniques for activity 1a • Model, tools or techniques for activity 1b	Any documents/ artefact of prototype of a particular phase is shown here
Phase 2	• Activity 2a • Activity .. • Activity 2n	• Model, tools or techniques for activity 2a • Model, tools or techniques for activity 2.. • Model, tools or techniques for activity 2n	Deliverable 2
Phase 3	• Activity 3a • Activity .. • Activity 3n	• Model, tools or techniques for activity 3a • Model, tools or techniques for activity 3.. • Model, tools or techniques for activity 3n	Deliverable 3
Phase 4	• Activity 4a • Activity .. • Activity 4n	• Model, tools or techniques for activity 4a • Model, tools or techniques for activity 4.. • Model, tools or techniques for activity 4n	Deliverable 4

As shown in Table 1, the SPM was decomposed into several phases and each phase was decomposed into several activities. To carry out activities of each phase, suitable tools, techniques, and models can be used which ultimately produce deliverable of each phase; documents such as software project management plan (SPMP), software requirement specification (SRS), software design document (SDD), software test plan (STP) and prototype of the software.

The details of SPM adopted were to be presented into a proper project management via various tools and techniques they could use for project planning, such as Gantt chart, program evaluation review techniques (PERT) chart, and work breakdown structure (WBS). The students were required to produce a Gantt chart by listing all important tasks and determine the predecessor and duration of each activity which use abstraction techniques of CT. Furthermore, they need to know the sequence of the tasks that need to be carried out throughout the project, depending on the SPM that they chose, which could be V model, RAD, RUP, prototyping and many more, which implies the algorithm concepts of CT. In addition, the illustration of the SPM needs to be included in their report, which depicts the abstraction aspects of CT. PERT chart is another tool that should be used to identify critical path which give the information on which tasks are critical and cannot be delayed.

Furthermore, the slack time of each task can be identified easily by using the PERT Chart, presenting abstraction concept of CT. Both Gantt Chart and PERT chart need to be constructed using suitable CASE tools such as Microsoft Project, GanttProject, GanttPro and be demonstrated to the coursemates as a hands-on mechanism.

Other than project management skill, modeling of software using different types of diagrams is crucial to illustrate different aspects of the software; the structural, functional and behavioral. The scope of their project is reflected by a use case diagram which is included in the SRS. The abstraction concept of CT is used as the students need to identify users (actors) and main functions of the software which is being developed. This diagram reflects the functional requirement of the software while the non-functional requirement refers to the quality aspects of the software which support the functionalities of the software. To explain important concept in constructing the use case diagram, a series of questions were asked to the students; who are users of the system (actors) and what do they do when use the system (use cases). Based on the given answers, the lecturer explained that the use case represents the functions of the system while actors are users of the system. The activity diagram, another important diagram in SRS to represent the sequence of functions throughout software, applied algorithm technique. The swim lane activity diagram not only use algorithm but also abstraction technique, to differentiate users who uses certain functions of the software.

In design phase, the four types of design; User interface, database, architectural and component design were highlighted in SDD. The selection of architectural design is based on the types of software that they develop mobile application, web-based system, game application, augmented reality (AR), virtual reality (VR), and the others. Pattern recognition is the CT technique which is applied here as the layout of the component; user interface, processing logic and database need be arranged accordingly so that they can work efficiently to deliver output of the software. For the database design, the entity relationship diagram (ERD) or class diagram, the abstraction technique was useful in identifying important entities and relationship among them. Another important diagram in SDD is sequence diagram or collaboration diagram which shows the flow of each use case, reflected the algorithm technique of CT.

The development phase required the students to use different software to develop different component of the software. An online forum was created as a medium for the students to discuss various software and framework to develop different types of software. The student-centric approach encourages cooperative learning among them where they have opportunity to discuss various aspects of development with the students from the other groups, A, B and C.

Such an approach is in line with Nasir and Sahibuddin's (2011) research findings indicating clear requirements, realistic estimations of the schedules and budgets, and effective project management were the critical success factors of software projects. In particular, managing a software development project using appropriate SPM is key to reducing the risk of failures (Paula, 2009). Given that no two projects are exactly alike, managing evolving requirements, expectations, and technologies can lead to unique innovations, such as agile practices. Although traditional engineering disciplines have been promoting "makers" as a new means of understanding the design process, software engineers have long recognized the need to determine the transition between analysis and design and between requirements and validation, and, more recently, between implementation and deployment.

According to Bell (2010), many of these skills cannot be measured using standardized tests. As such, the measurements of such skills can be based on students' self-evaluation, self-reflection, and constructive feedback to highlight their strengths and to improve their interactions with one another. Through reflection, they could gauge how well they had collaboratively worked in a group and how well they had contributed, negotiated, listened, and welcomed other group members' ideas. Likewise, they also had to self-evaluate their own projects, efforts, motivations, interests, and productivity levels.

3. Result and Discussion

The three groups, namely Group A, Group B, and Group C, consisted of 25.7%, 35.7%, and 38.6 % of the selected students, respectively. These students who enrolled in this course were those in their fifth, sixth and seventh semesters, the percentages of which were 24.3%, 28.0%, and 72.9%, respectively. Responses to most of the questions were rated along a 5-point Likert-type scales, ranging from '1' (Least) to '5' (Most). The first question asked about their enjoyment of the course as follows: "How much had you enjoyed the course?" All of them indicated that they moderately and highly enjoyed the course, registering percentages of 35.7% and 64.3 %, respectively. The main question sought to determine students' knowledge and skills in SE in terms of software project planning, software requirement and specification (modelling), software design, development, testing, software process modelling, and presentation skills, before and after PBL-based learning. The responses to the main question were also rated along -point Likert-type scales, namely '1' (very poor), '2' (poor), '3' (fair), '4' (very good), and '5' (excellent). Students need to acquire such important skills through various courses that are offered in the software engineering program, some of which are compulsory and others optional. In this course, the lecturer recalled the contents of such courses using CT techniques which being performed with a series of questions, before the students proceeded with their projects. Also, appropriate tools and techniques were demonstrated to them in this course.

The analysis of students' answers showed that the levels of their project planning skills were moderate based on their responses that ranged with 17, 9, 11, and 4 students reporting their skills to be very poor, good, very good, and excellent, respective. Surprisingly, none of the students claimed that their skills were very poor after learning. Promisingly, more than three-quarters (75%) of the students indicated that their project planning skills were more than average, followed by 35 and 18 students who indicated that their skills were indicating their skills very good and excellent, respectively.

The same analysis showed that the levels of students' software requirement modelling skills were higher than those of their project planning skills, as none of them indicating that their skills in software requirement modelling were very poor. Also, only three students indicated they were poor in such skills. By contrast, many of the students reported that their software requirement modelling skills were fairly good and excellent, each represented equally by 16 students. Equally impressive, 35 students indicated that their skills in software requirement modelling were very good. All aspects of requirements, including functional requirements and non-functional requirements, were discussed in the course before they started gathering the requirements of the software, together with suitable models, tools, and techniques.

Furthermore, the results showed that 71.4 % of the students indicated that their levels of software design skills after the SE Project course were very good and excellent were, far exceeding their levels of such skills before the course where only 24.2 % of the students made such a claim. Moreover, none of them indicated their levels in such skills were very poor after the course, which contrasted with the findings before the course where three (3) students indicated that their levels in such skills were very poor. In addition, a majority of the students said that their levels of software design skills were satisfactory and poor before the course, with 24 and 26 students claiming that their skills were very poor and fairly good, respectively. However, the same students indicated that had made significant improvements after the course, with 33 and 17 students claiming that their levels of such skills were very good and excellent, respectively. In this course, they learned four types of design architecture, data, user interface, and components that are highly emphasized in the Software Design Document (SDD).

Before the course, 34%, 40%, and 25% of the students claimed that their levels of development skills (which are related to coding skills) were fairly good, very poor and poor, and very good and excellent, respectively. After the SE Project course, none of the students indicated that their levels of these skills were very poor level, with only three (3) (4.2% of the students) claiming their skills to be poor. Overwhelmingly, 95.8 % of the students reported that their development skills were fairly good, very good, and excellent, which stood at 25.7%, 48.6%, and 25.7%, respectively, signifying a significant improvement in such skills. In this course, they learned several revised programming languages together with the compatibility of different languages. Also, several languages for the development of different components of a software application, such as database, and client- and server-side programming were discussed prior to the development process. Additionally, they learned the Integrated Development Environment (IDE) and Computer Aided Software Engineering (CASE) tools to determine the suitability of each tool for the different types of software, such as mobile applications, web-based systems, games, and augmented reality applications.

Testing was one of the software development activities that they performed. Before the course, 17%, 12.8%, and 4.2% of the students' scores in software testing were average, very good, and excellent, respectively. Remarkably, they made significant improvements after the course, with 51.4% and 12.9% of attaining very good and excellent testing scores, respectively, clearly indicating the positive impact of the SE Project course. In this course, the discussion focused on the methods, types, and techniques of software testing.

Likewise, the students also made significant improvements in the SPM. Prior to the course, 2.86 % and 20% of the students indicated that their levels of skills were very poor and very good, respectively. After the course, none of the students indicated that their skills were very poor. Moreover, 45.7%, 31.3%, and 18.55 of the students reported that their methodological skills were very good, excellent, and fairly good, respectively.

After the course, students' presentation skills improved significantly, as evidenced by 92.9% of them attaining scores above the average. Specifically, 17.1%, 40% and 35.7% of the students stated that their levels of this skill were fairly good, very good, and excellent, respectively. Furthermore, none of them indicated that they had very poor presentation skills. By contrast, 3.3% of the students indicated that they had very poor skills in presentation before the course. In the presentation session, they were required to present the progress of their projects by highlighting each of the milestones.

Table 2 summarizes students' pre- and post-test mean scores of the seven skills related to SE knowledge. Seemingly, the course had a profound impact on the improvement of such

skills, given the substantial differences between the pre- and post-test mean scores of such measures. For example, before the course, students' mean scores of project planning and software process model skills were 2.56 and 2.89, respectively. After the course, their mean scores of the same measures were 3.76 and 3.87, respectively, signifying remarkable improvements in these two skills.

Table 2: Students' pre- and post-test mean scores of the six aspects of SE knowledge and skills

No.	SE knowledge and skills	Before	After
1.	Project planning	2.56	3.76
2.	Software requirement & specification (modelling)	2.73	3.67
3.	Software design	2.84	3.64
4.	Development	2.82	3.64
5.	Testing	2.64	3.42
6.	Software Process Model	2.89	3.87
7.	Presentation	2.78	3.76

Furthermore, the paired samples *t* test was used to determine whether the differences between pre- and post-test mean scores of the seven skills of SE knowledge, the results of which are summarized in Table 3.

Table 3: The results of dependent samples *t* test

	N	Overall SE Skills		
		Mean	Std. Deviation	Std. Error Mean
Before the PBL course	70	2.75	0.2270	0.0442
After the PBL course	70	3.68	0.1413	0.0534

As shown, the overall mean scores of students' SE skills before and after the course were 2.75 and 3.68, respectively. There was a significant difference in the scores before the PBL-CT course (M=2.75, SD=0.227) and after (M=3.68. SD=0.1413). A repeated-measures t-test found this difference to be significant with t (6) = 16.724, $p < 0.0001$ as shown in Table 4.The value of *t* is 16.723572 and the value of *p* is < .00001.

Table 4: The results of paired samples *t* test

	Paired difference				
			Std. Error Mean	95% Confidence Interval of the Difference	
	Mean	Std. Deviation	2.75	Lower	Upper
Pair 1 Before- After	0.92857	.14690	0.5552	1.06444	0.79271

	t	df	Sig, (2-tailed)
Pair 1 Before- After	16.724	6	.000

With a significance value of less than 0.05, the difference between the pre- and post-test mean scores were found to be extremely statistically significant. Thus, the null hypothesis, there was no significant difference in the level of SE skills and knowledge before and after the software engineering project class, was rejected.

This finding showed that the students had made a significant gain in the overall SE skills, suggesting that PBL with CT technique is an effective approach to help SE students to develop strong knowledge and skills in various aspects of SE. This finding is consistent with that of a study by Souza et al. (2019) who found PBL helped increase students' understanding of several topics of software development. Being a student-centered approach, PBL is an efficacious learning method to help students develop holistic skills by bridging the theoretical and practical aspects of SE (Peres & Rubio, 2020). Clearly, the successful implementation of PBL would entail CT with its four main techniques, decomposition, abstraction, algorithm, and pattern recognition. In addition, cooperative learning could be fostered through the combination of its approaches, student-centric, hands-on, and inquiry-based.

Interestingly, Kokotsaki et al. (2016) carried out an intensive study that focused on the effectiveness of PBL on student learning that covered a broad spectrum of educational levels (elementary, primary, secondary, and tertiary levels) and fields (history, science, engineering, and mathematics). Their study showed that PBL helped improve students' motivation in learning. Of particular interest was the impact of such a learning method on assisting students in Britain to develop strong understanding of science and technological concepts.

Additionally, this study aimed to investigate students' perceptions of the relevant activities of a CT approach used in the course before performing their practical tasks based on PBL, which are summarized in Table 4. All groups were not only required to involve and present their progress as scheduled by the lecturer, but also give inputs to their classmates' progress. As shown, most of the students indicated the learning activities that they performed had greatly benefited them, as evidenced by many of their responses to the questionnaire items falling under the scales of '4' and '5'. By contrast, none of the students' responses fell under the scales of '1' and '2'.

Table 4: Students' perceptions of the course activities in SE course using cooperative learning

Activities	Scale				
	1	2	3	4	5
Video and discussion of current trend of software development and different types of software (pattern recognition)	0	0	6	31	33
Video and discussion of Software Process Model (decomposition)	0	0	7	28	35
EdPuzzle about the concepts and theories of SE (abstraction)	0	0	6	33	31
Video on software requirement	0	0	6	26	38
Presentations of software design using various tools (abstraction)	0	0	5	23	42
Web-server hosting demonstration	0	0	9	35	26
Project planning demonstration (Gantt chart, PERT chart) (decomposition and algorithm)	0	0	3	24	43
Deliverable presentation of modelling (use case (abstraction)/ activity diagram (algorithm)	0	0	3	22	45
Deliverable presentation and discussion (sequence and class diagram) (algorithm& abstraction)	0	0	4	23	43
Presentation and discussion of project deliverables	0	0	3	26	41
Forum on software development tools (pattern recognition)	0	0	10	23	37

Peer evaluation of software prototype	0	0	4	23	43

Figure 1 shows the distribution of students' responses to questions asking them about the four aspects of the PBL course.

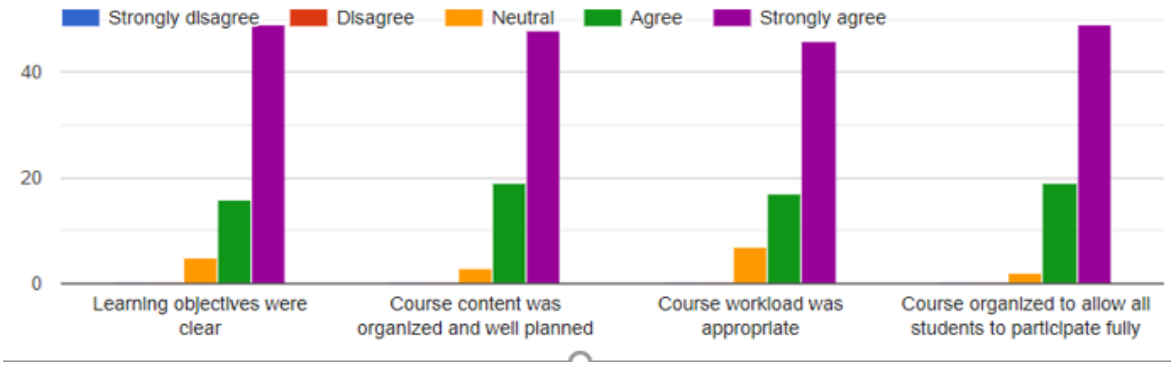

Figure 1: The students' opinions on the four aspects of the PBL course

As shown, most of the students strongly agreed with the questionnaire items' statements relating to the four aspects of the PBL-CT course, with none of them disagreeing or strongly disagreeing. Specifically, 49 students (70%) strongly agreed with the first questionnaire item's statement that the learning objectives are clear. By contrast, 16 participants (22.8%) responded that they agreed with such a statement, and only five (5) participants (7.14%) remained neutral with the statement. For the second questionnaire item's statement, 'The course content is well-organized and well- planned', their responses were about the same as those of the first questionnaire item. It was observed that 48 participants (68.6%) responded that they strongly agreed with the above statement, followed by 19 participants (27.1%) and three (3) participants (4.3%) agreeing and remaining neutral with the statement, respectively.

The third questionnaire item's statement was about the appropriateness of the course workload. Forty six (46) students (65.7%) responded that they strongly agreed with the statement. The remaining 17 students (24.3%) and seven (7) students (10%) indicated that they agreed and remain neutral with the same statement, respectively. The last questionnaire item concerned the organization of the course to foster full participation to which 49 respondents (70%) and 19 participants (27.1%) responded that they strongly agreed and agreed with it, respectively. By contrast, only two (2) students (2.85%) wanted to remain neutral.

The last section of the survey consisted of an optional open-ended question to which students could freely comment on any aspects of the course. The analysis of the comments made by 10 students showed that they managed to gain strong understanding of project deliverables, such as SPMP, SRS, and SDD documentation. For example, a student indicated that she was able to learn to produce a complete set of documents perfectly based on her comment as follows: 'This course helps me to prepare the SE documentation'.

They also claimed that their lecturer's comments and feedback for each deliverable were very helpful, as indicated by one student's comments as follows: 'There are a lot of new things that I have learned in this course. I loved it when my lecturer and friends shared her thoughts on various tools to use in software development. It gave me the courage to look for new things to learn and use them in my project'.

Another insightful comment could be gleaned from another student's comment as follows: 'I have learned a lot about CASE tools that could be used in the different phases of

an SDLC, such as Gantt project, Microsoft Visio, MySQL Workbench, and others. Other students' comments were also highly revealing as follows: 'Most importantly, I could learn about project planning in addition to modelling and development' and 'I received better guidance on project management that helped me to understand various terms of the technology'. Likewise, another comment made by another student was illuminating as follows: 'This course helped simplify all that we have learned from Semester 1 to Semester 7'. Arguably, the brainstorming or sharing sessions and the teaching approach with examples helped to students to learn more efficaciously.

Another important aspect that was commented on by the students was the software requirement specification through the modelling of the actual software to be developed. Not only their technical skills had improved, their soft skills had also improved through this PBL-CT course, which was supported by the lecturer's guidance. Given that the progress of every group's project had to be presented in the course, many students were able to learn about and avoid the mistakes made by other students. They indicated that the presentation of project progress helped improve not only their presentation skills but also their confidence. They also claimed that peer-to-peer activities, such as peer assessment, were very helpful to encourage them to do the best, which is exemplified by several students' comments as follows: 'The presentation of each project deliverable helped students to learn from their peers' and 'In the presentation, everyone had to provide a justification for his/her answer, and this made me a better analyzer'. Students' participation in each phase of software development was also monitored and evaluated to reveal their level of engagement and critical perspective, which are highlighted in some of their comments as follows: 'Student could participate fully' and 'I became bolder to speak and more critical in exploring new knowledge, as there is no spoon-feeding in this course'.

Extensive communication was also emphasized in the course to help students gain strong communication skills. In this regard, the students indicated that they had made strong improvements in such skills, which are best exemplified by some of their comments as follows: 'I learned to communicate in a very good way with all my course mates, and I am no longer afraid to ask questions from the lecturer or anyone else'. Equally important, the students opined that this course was highly effective in improving their knowledge, which are revealed in the following comments: 'I found it was easy to participate in this course compared to other courses, although this course was quite tough, 'Through this course, I am now able to organize a team project properly', and 'This course was the best among other courses because the lecturer cared to repeat and relate various aspects of SE methods, tools, and techniques that we had learned previously'.

The students also expressed their positive feelings about this innovative learning course, which can be discerned from some of their comments as follows: 'I love this course so much, and I keen to learn more. I am grateful and really appreciate for all the knowledge and new things in SE that I had learned in this course', and 'I would like to express my gratitude for such a wonderful preparation of the course that has helped prepare me for the coming industrial training'

Inevitably, like any other courses, there are also some drawbacks in this course, which have to be properly addressed to make it more effective for future uses in terms of the size of a project group and the target students. There was a suggestion to have more team members per group, preferably more than three, which was the case in this study. With more team members, more ideas and insights can be shared to improve the learning efficacy. Also, selecting suitable students to attend the course needs to be carefully considered based on

learning semesters, which was emphasized by some of the students who made the following suggestion: 'I think, it is better for students in Semester 6 to take this course so that they can prepare for their final projects, not like us who are enrolled in Semester 7' and 'If students can take this course in early semesters, they can avoid the chaos in producing their final year projects. Evidently, these comments suggest that the timing of the course needs to be revised by allowing students in early semesters, ideally students in their sixth semester, to take the course.

4. Conclusions

In this study, the implementation of a PBL approach with the integration of CT technique to support the learning of software engineering provided students with a real-world project experience. The students managed to gain valuable experiences by working as a software engineer in addition to working in a team via cooperative learning. Various essential skills that a software engineer needs to have, such as project management, requirement specification, software design, development, and testing, were applied in this course with the PBL-CT approach. This study showed students' knowledge and skills, such as technical, management, and communication skills, made significant improvements completing the course, encompassing cognitive, practical and affective domains of learning. In this project, students' reasoning and thinking skills were nurtured through problem-solving activities that they performed to accomplish their projects, such as planning the projects, modelling the requirements, designing the system, developing various components of the software, and implementing the software using a programming language. Practical skills, especially in using various CASE tools throughout the development of the software, were enhanced in this course, and all deliverables were presented and assessed according specified milestones. Students were given freedom to choose software tools for project planning, diagramming, and programming, allowing them to try out various software applications before choosing the right software in producing project deliverables, various documents, and the prototype of the software.

Students were found to be highly engaged in this course, which were gauged during the presentation of project deliverables, in addition to peer assessments. However, some students faced difficulties in rating their friends' work as the assessment rubrics provided to them lacked details. In the future, the current rubrics need to be itemized in detail to guide peer-assessment, cooperative learning. Also, problems associated with a lack of time to complete projects and lacks of cooperation among some students were unavoidable, which impeded smooth project delivery. On a positive note, however, most students were observed to be enthusiastic and receptive to learn important aspects of SE using the PBL-CT approach that stressed on solving real-world problems. Overall, the results of this study suggest that the PBL-CT approach is highly effective in helping students to develop SE skills deemed essential to a competent a software engineer. As demonstrated, such an approach can support the teaching and learning process of SE by striking a solid balance between theoretical and practical aspects, enabling students to retain more information, enhance their knowledge, foster collaboration, and nurture their critical thinking skills. Clearly, the successful implementation of PBL-CT would entail a combination of approaches, such as

student-centric, hands-on, and inquiry-based approaches which indirectly reassure cooperative learning.

The presented findings provide a practicable approach of PBL-CT for future teaching to novice educators. The aim to evaluate the effectiveness of PBL with CT techniques in a Software Engineering Project Course comes with guidelines for teachers to help them design a series of lessons aimed at developing SE skills and knowledge.

References

Bell, S. (2010). Project-based learning for the 21st century: skills for the future. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas, 83*(2), 39–43.

Blumenfeld, P.C., Soloway, E., Marx, R.W., Krajcik, J. S., Guzdial, M., & Palincsar, A. (1991). Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist, 26*(3-4), 369–398.

Gary, K. (2015). Project-based learning. *Computer, 48*(9), 98–100.

Ghezzi, C., & Mandrioli, D. (2005). The challenges of software engineering education. *Proceedings of the 27th International Conference on Software Engineering - ICSE '05*. https://doi.org/10.1145/1062455.1062578

Gillies, R' M., & Boyle, M. (2011). Teachers' reflections of cooperative learning (CL): A two-year follow-up. *Teaching Education, 22*(1), 63-78.

Hazzan, O., & Dubinsky, Y. (2007). Why software engineering programs should teach agile software development. *ACM SIGSOFT Software Engineering Notes, 32*(2), 1. https://doi.org/10.1145/1234741.1234758

Kokotsaki, D., Kokotsaki, V. & Menzies, W. (2016). Project-based learning: A review of the literature. *Improving Schools, 19*(3).

Mittal, M., & Sureka, A. (2014). Process mining software repositories from student projects in an undergraduate software engineering course. In *Companion Proceedings of the 36th International Conference on Software Engineering*, May (pp. 344-353).

Pérez, B., & Rubio, Á.L. (2020). A project-based learning approach for enhancing learning skills and motivation in software engineering. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3328778.3366891

Saad, A. (2020). Students' computational thinking skill through cooperative learning based on hands-on, inquiry-based, and student-centric learning approaches. *Universal Journal of Educational Research, 8*(1), 290-296.

Santos, S.C., da Conceição Moraes Batista, M., Cavalcanti, A.P.C., Albuquerque, J.O., & Meira, S.R.L. (2009). Applying PBL in software engineering education. *2009 22nd Conference on Software Engineering Education and Training*. https://doi.org/10.1109/CSEET.2009.39

Shaw, M. (2000) Software engineering education: A roadmap. In Proceedings of the Conference on The Future of Software Engineering, May (pp. 371–380). https://doi.org/10.1145/336512.336592

Sindre, G, Giannakos, M., Krogstie, BR. Munkvold, R, & Aalberg, T (2018). Project-based learning in IT education: Definitions and qualities. *Uniped, 41*(02), 147-163.

Souza, M., Moreira, R., & Figueiredo, E. (2019). Students perception on the use of project-based learning in software engineering education. *Proceedings of the XXXIII Brazilian Symposium on Software Engineering - SBES 2019.* https://doi.org/10.1145/3350768.3352457

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J.T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education, 14*(1), 1–16.

Yadav, S.S., & Xiahou, J. (2010). Integrated project based learning in software engineering education. *2010 International Conference on Educational and Network Technology.* https://doi.org/10.1109/icent.2010.5532120