Journal of Information Technology Education: Innovations in Practice

An Official Publication of the Informing Science Institute
InformingScience.org

JITEiip.org

Volume 17, 2018

COLLABORATIVE APPROACH IN SOFTWARE ENGINEERING EDUCATION: AN INTERDISCIPLINARY CASE

Aileen Joan Vicente*	College of Science, University of the Philippines Cebu, Cebu City, Philippines	aovicente@up.edu.ph
Tiffany Adelaine Tan	School of Management, University of the Philippines Cebu, Cebu City, Philippines	tgtan@up.edu.ph
Alvin Ray Yu	College of Science, University of the Philippines Cebu, Cebu City, Philippines	aoyu3@up.edu.ph

*Corresponding Author

ABSTRACT

Aim/Purpose	This study was aimed at enhancing students' learning of software engineering methods. A collaboration between the Computer Science, Business Management, and Product Design programs was formed to work on actual projects with real clients. This interdisciplinary form of collaboration simulates the realities of a diverse Software Engineering team.
Background	A collaborative approach implemented through projects has been the established pedagogy for introducing the Software Engineering course to undergraduate Computer Science students. The collaboration, however, is limited to collaboration among Computer Science students and their clients. This case study explored an enhancement to the collaborative approach to project development by integrating other related disciplines into the project development framework; hence, the Interdisciplinary Approach.
Methodology	This study adopted the case method approach. An interdisciplinary service innovation activity was proposed to invite other disciplines in the learning process of the computer science students. The agile methodology Scrum was used as the software development approach during project development. Survey data were collected from the students to establish (a) their perception of the interdisciplinary approach to project development; (b) the factors that influenced success or

Accepted by Editor Athanassios Jimoyiannis | Received: March 16, 2018 | Revised: May 8, May 19, 2018 | Accepted: May 22, 2018.

Cite as: Vicente, A. J., Tan, T. A., & Yu, A. R. (2018). Collaborative approach in software engineering education: An interdisciplinary case. *Journal of Information Technology Education: Innovations in Practice, 17,* 127-152. https://doi.org/10.28945/4062

	failure of their team to deliver the project; and (c) the perceived skills or knowledge that they acquired from the interdisciplinary approach. Analysis of data followed a mixed method approach.
Contribution	The study improved the current pedagogy for Software Engineering education by integrating other related disciplines into the software project development framework.
Findings	Data collected showed that the students generally accepted the interdisciplinary approach to project development. Factors such as project relevance, teamwork, time and schedule, and administration support, among others, affect team performance towards project completion. In the case of the Computer Science students, results show that students have learned skills during the experience that, as literature reveal, can only be acquired or mastered in their future profession as software engineers.
Recommendations for Practitioners	The active collaboration of the industry with the University and the involvement of the other related courses in teaching software engineering methods are critical to the development of the students, not only in learning the methodology but also as a working professional.
Recommendation for Researchers	It is interesting to know and eventually understand the interactions between interdisciplinary team members in the conduct of Software Engineering practices while working on their projects. More specifically, what creative tensions arise and how do the interdisciplinary teams handle the discourse?
Impact on Society	This study bridges the gap between how Software Engineering is taught in the university and how Software Engineering teams work in real life.
Future Research	Future research is targeted at refining and elaborating the elements of the interdisciplinary framework presented in this paper towards an integrated course module for Software Engineering education.
Keywords	software engineering education, interdisciplinary learning, collaborative approach

INTRODUCTION

Software Engineering is one of the most practical courses in Computer Science and Engineering. The prescribed course implementation will always require the development of software alongside the introduction of Software Engineering methodologies, techniques, and tools in lecture classes. This study explored a case of a collaborative approach to software project development in an interdisciplinary environment.

Many Software Engineering Education case studies focus on collaboration among Computer Science or Software Engineering students (Giraldo, Collazos, Ochoa, Zapata, & de Clunie, 2010; Kirti, Sureka, & Varma, 2015) or collaboration between Computer Science students and real clients (Chase, Oakes, & Ramsey, 2007; Chen & Chong, 2011; Flener, 2006). These case studies agree that students get the maximum appreciation and learning of Software Engineering methods and techniques when applied to real-world projects. These studies also show that students learn from their peers and from real-world challenges in a collaborative environment. For these studies however, the collaboration is limited to peers with the same educational background. The notion of learning in Software Engineering from other disciplines is rarely studied. Beard (2014) argued that sharing knowledge between Software Engineering and other disciplines can help improve Software Engineering practices. Therefore, the inclusion of non-Computer Science students, with different but related course requirements, to the project teams makes this case an interesting study.

This interesting case of interdisciplinary environment exists in software development teams today as well. As software engineering practices become agiler, software engineers need to shift from software orientation to a more user-centric paradigm. Thus, there is a need for software engineers to become more people oriented. An increased collaboration between stakeholders and development team is necessary. Because software engineers tend to be "abysmal with people" (Capretz, 2003), organizations create diverse teams composed of both engineering and non-engineering professionals (Cooper, n.d.). Interestingly, this industry environment is not simulated in the project development experience of our students. In order to bridge the gap, this study enhanced the collaborative approach to Software Engineering education by inviting other disciplines to work with Computer Science students in the development of a software project.

This paper presents this interdisciplinary case and is organized as follows. The paper begins with the presentation of the key pedagogies applied in this study, that is, the Collaborative and Interdisciplinary approaches. From this, the study's proposition is presented using a framework for an interdisciplinary approach to software project development. The research methodology used is detailed including the presentation of this research's case context. The results and discussions sections present the findings from the data collected and their subsequent interpretations. Finally, the paper concludes by synthesizing the results vis-à-vis the theoretical underpinnings of the study. Recommendations for future work are presented as well.

LITERATURE REVIEW

One of the challenges in academic education of professional courses is the continuous revision of the course syllabus to meet the growing demands of the profession and the industry. Software Engineering is one of those professional courses that is required to adapt continuously. Most faculties acknowledge that there are gaps between how Software Engineering is being taught in the classroom and how Software Engineering teams work in real life. It is therefore imperative that the academe continues to design the curriculum to bridge these gaps.

Two related pedagogies are presented in this study – Collaborative and Interdisciplinary. Each has its own hallmark in the field of education. The collaborative approach has been the primary approach to teaching Software Engineering through projects. This study improved teaching and learning Software Engineering by integrating an interdisciplinary aspect. In doing so, we move a step closer to bridging academe training and industry practices.

COLLABORATIVE APPROACH IN SOFTWARE ENGINEERING EDUCATION

Research in Software Engineering Education suggests that a more practical delivery of the course through projects is favorable. This is where a collaborative approach to teaching Software Engineering comes in. Collaborative learning is an umbrella term for a variety of methods in education that involve "the joint intellectual effort by students or students and teachers" (The University of Sydney School of Education and Social Work, 2018). This methodology covers activities such as collaborative writing and group projects. The collaborative work in Software Engineering Education is one that requires students to work together to explore and develop a software solution to a problem. According to Blaschke (2012), these types of educational strategies are often driven by an emphasis on providing students with the skills and attributes to become "self-determined" and "highly autonomous" life-long learners. This also means that lessons learned during their collaborative experience are deeply ingrained more than the lessons learned in the classroom.

Most Software Engineering Education case studies presented different ways of collaboration among Computer Science students (Giraldo et al., 2010; Kirti et al., 2015) using various frameworks and tools. Some presented results of collaboration between students and real clients (Chase et al., 2007; Chen & Chong, 2011; Flener, 2006). Mead (2015) acknowledged this need for university-industry collaboration in order to advance Software Engineering education. She proposed a strategy in the

form of industries serving as real clients to the students to provide for practical industry problems, to provide direction as to the technology to be used and to help review deliverables.

Various researchers have also presented different strategies for teaching Software Engineering. For instance, Gannod, Burge, and Helmick (2008) studied the use of inverted classroom and Reichlmayr (2005) utilized blended learning approach. Stein (2002) experimented on group size (small or large groups) in capstone and Software Engineering courses. Stettina, Zhou, Bäck, and Katzy (2013), on the other hand, introduced the intensive coaching of teams and the notion of team routines. Chen and Chong (2011) also introduced the meetings-flow approach as a process to improve software engineering practice. For most of these researchers cited, collaboration among software engineering students is emphasized.

Collaborative learning has been generally employed in other Computer Science courses as well especially in programming classes. For example, Bower and Richards (2006) designed and implemented a collaborative framework for object-oriented programming that gained positive response from students. They found that better performance and increased student satisfaction demonstrate that programming in teams can operate successfully and improve educational outcomes.

Interdisciplinary Approach in Software Engineering Education

Interdisciplinary teaching is a method used to teach a topic across different curricular disciplines. McCuskey and Conaway (1995) said that the approach is "really an old friend under a new-old name" (p. 395). Accordingly, interdisciplinary teaching is already practiced by some teachers when they try to make concrete the abstract concepts using other related disciplines as context. By doing so, the students can grasp these abstract concepts easily. In addition, the way that the curriculum is developed can inherently be interdisciplinary. An example is presented by Rick Oches (2012) from Bentley University when he integrated the topic on ethanol to teach sustainability issues. The module he created on ethanol tied together other disciplines such as Chemistry, Ecology, Economics, Engineering, Environmental Science, Geology, Political Science, and more. In the course design, the students understood the production of ethanol by being tasked with making it a part of an environmental chemistry course. In addition, the students are also tasked to perform a cost-benefit analysis for ethanol production and consumption in a microeconomics course. And finally, they learn about the legislative process involved in energy production and use this in a public policy course. Clearly, this interdisciplinary approach induces a deeper understanding of ethanol and its production.

The hallmark of an interdisciplinary learning is on the cognitive development of the students participating in the interdisciplinary environment (Repko as cited in Goldsmith, Hamilton, Hornsby & Wells, n.d.). Specifically, Repko (as cited in Goldsmith, Hamilton, Hornsby & Wells, n.d.) identified four cognitive abilities that are cultivated in interdisciplinary learning. First, is the Perspective-taking Technique. This is the capacity of the learner to understand a topic or problem from multiple viewpoints. This understanding leads to an appreciation of different disciplines and their perspective on the topic. Second, is the Development of Structural Knowledge. This knowledge is a combination of Declarative Knowledge or the factual information and Procedural Knowledge or the process-based information used to solve complex problems. The third cognitive ability pertains to Integration of conflicting insights. Learners become creative when faced with various and maybe conflicting perspectives to a problem. As a result, new and better solutions can be created. The last cognitive ability is Interdisciplinary Understanding. This entails that learners see the situation through different lenses and recognize how each perspective affect the other.

Interdisciplinary learning has been applied in various ways in the Computer Science program. One implementation is seen in the field of research. As an example, Simon Fraser University implemented the Modelling of Complex Social Systems (MoCSSy) Program in 2008. The program allowed students from different disciplines to work on research problems within the program's five research themes – culture, society and human behavior, communication, computation and technology, and

health (Giabbanelli, Reid & Dabbaghian, 2012). With this approach, Giabbanelli et al. (2012) found that Computing science majors and non-majors have learned to appreciate working with each other and proposed solutions that were recognized by the academic community. Similar interdisciplinary research programs have also been implemented at George Mason University for the Ph.D. in Computation Social Science and Carnegie Mellon for Ph.D. in Computation, Organizations, and Society (as cited in Giabbanelli et al., 2012).

An interdisciplinary approach can be implemented in course design as well. For example, the Multimedia Information Systems at the University of Otago was designed to be an interdisciplinary program taught by faculty from both the Design Studies and Information Science Departments (Wong, McGuire, & McDonald, 1996). They argued that their approach to Multimedia education is closer to the way real-world titles are produced. They justified that "multimedia production is by necessity a multi-disciplinary effort, requiring the talents of both creative and technical professionals" (Wong et al., 1996).

O'Leary and Azadegan (2005) also published a similar interdisciplinary course for scientific modeling and simulation. The course was participated by Computer Science, Science and Math majors. They found that despite the differences in background knowledge and experiences, the students liked the challenges. These challenges are introduced by deficiencies in their background in mathematics, programming and science, the complexity of mathematics involved, and the difficulty of the course projects. Their results suggested that learning something practical made the course interesting. As a result, the course has been well received by the students and many of them felt they learned much.

The research of O'Leary and Azadegan (2005) implemented interdisciplinarity using problem-based learning. Nikitina (2006) classified this type of implementation as a potential strategy for interdisciplinary teaching. Problem-solving in this respect involves enlisting the knowledge and modes of thinking in several disciplines to address real-life problems that take more than one discipline to solve (Nikitina, 2006). When applied to team-based problem-solving, interdisciplinary learning, therefore, can be seen as a form of collaboration among different disciplines.

In the case of Software Engineering Education, however, this collaboration with other disciplines in the delivery of the course is rarely studied. As cited in the literature, the collaboration in most case studies is limited to collaboration among computer science students and their clients. A reason would be that learning the software engineering skills themselves are already "quite a bit to master" (Jaccheri & Sindre, 2007) and an interdisciplinary environment can add more complexities to the project as it is. Despite the complexity, Jaccheri and Sindre (2007) explored this interdisciplinary environment for Software Engineering Education by creating the Expert in Team (EiT) course. The course was enrolled in by Software Engineering students and students in many other various programs. The course encouraged creativity by allowing interdisciplinary teams in so-called Software Villages (themed villages) to identify their problems and goals and to develop solutions within their village teams. They reported that such an interdisciplinary course gave students learning outcomes that are entirely different from what they get from more traditional software engineering team projects, in particular, concerning interdisciplinary skills and self-insight.

The problem however in EiT is that it had an open-discipline structure. The course was structured as such in order to foster creativity among students with an emphasis on self-reflection as a learning outcome. Students taking the course are presumed to have already taken the prerequisite Software Engineering courses. Hence, Software Engineering methods and practices, as a learning outcome, are not emphasized.

This case study differed from the work of Jaccheri and Sindre (2007) in the structure of the interdisciplinary framework. While Jaccheri and Sindre (2007) introduced an open-discipline environment, this study presented a multi-disciplinary structure that mimics the typical composition of Software Engineering teams based on roles. The justification for this environment follows from Lave's (1988) learning theory of Situated Learning (as cited in Ellis, Demurjian & Naveda, 2009). Situated Learning

is concerned with the environment where active learning takes place. In this learning theory, knowledge is situated as a product of the activity, context, and culture in which it is developed. Therefore, in order to be effective, the project environment should be as authentic as possible. Ellis et al. (2009) cited several applications of this theory in Software Engineering education that incorporates aspects of realism into the class project such as in the work of Hayes (2002) when they used an industrial participant to play the role of a customer and Favela and Pena-Mora (2001) when they studied large, distributed teams across geographical locations.

Another differentiation is seen in this case's learning outcome. Jaccheri and Sindre (2007) emphasized on self-reflection as a learning outcome. This study, on the other hand, focused on enhancing students experience to achieve a richer, deeper understanding of Software Engineering methods and practices.

Rus and Lindvall (2002) concluded that interdisciplinary problem-solving working groups are beneficial because "this will transfer the knowledge from one discipline to another, as well as provide solutions to interdisciplinary problems in decreased time". Interdisciplinary learning, therefore, affords a different kind of learning as it allows "acquiring new insights on what students have learned by applying it to new contexts" (Jaccheri & Sindre, 2007). It is therefore interesting to understand how these interdisciplinary working groups should interact in a classroom project setting to achieve this particular kind of learning.

Why Interdisciplinary Approach Is Timely and Relevant

In the previous section, the interdisciplinary approach was presented in the light of Situated Learning Theory. This section augments to the study's theoretical foundation by taking a look at diversity in real-world software engineering teams.

The concept of bringing diversity in software development teams has since piqued the interest of researchers in the attempt to improve team performance (Gottschalk & Solli-Sather, 2007). A review of 40 years of diversity research by Williams and O'Reilly (1998) concluded that there were no consistent main effects between diversity and organizational performance. According to them, instead of arguing that diversity positively affects team performance, certain mediating variables between diversity and performance may exist and be studied. To this end, Liang, Liu, Lin, and Lin (2007) explored the relationship of conflict as a mediating variable between diversity and performance. They concluded that Knowledge Diversity, interestingly, in contrast to their hypothesis, significantly increases task conflict. Task conflict, on the other hand, positively affects team performance. It follows then that Knowledge Diversity can positively affect team performance. Liang et al. (2007) further characterized Knowledge Diversity as the differences in education and experience. This goes to say that members with differing experiences and educational backgrounds may provide a variety of viewpoints, influence task conflict, and help improve the quality of decision making. These factors may save mission-critical projects at critical points in the project (Liang et al., 2007).

TechBeacon (www.techbeacon.com), a digital hub created for and by software engineers and IT professionals, published a number of articles written by Software Engineering professionals on the benefits of diversity in software engineering. Diversity is not limited to gender or ethnicity, as presented in most diversity and inclusion reports made by IT organizations. Real and meaningful diversity requires a collection of individuals with unique perspectives coming from their backgrounds, knowledge, past experiences and environments (Cooper, n.d.). While Cooper made this generalization in the context of Software Testing, the same cannot be any different in other phases of software engineering, such as in Requirements Engineering.

It is therefore appropriate that Software Engineering education inculcates the interpersonal skills necessary to make team diversity work. This industry need justified the study's practical relevance.

Research Objectives

The literature review showed that the approach to Software Engineering education through projects has been limited to collaboration between Computer Science students. Working with non-Computer Science students on software engineering projects is rarely employed, especially one that mimics professional software engineering teams. With the goal of enhancing student learning, this study explored the application of the interdisciplinary approach to software project development. This interdisciplinary approach, interestingly, mimics the composition of software engineering teams.

This case study explored the ways to improve the delivery of Software Engineering through projects and consequently, the students' learning. This exploration is guided by the following research questions:

RQ1. What factors strongly influence a successful software project in an interdisciplinary project environment?

RQ2. What software engineering skills or knowledge are enhanced through the interdisciplinary approach?

METHODS

Methodology

This study adopted an exploratory, single-case case study methodology (Yin, 2009). As the study's context, the implementation of the Interdisciplinary Service Innovation Program (ISIP) for one semester is studied. ISIP was participated in by students enrolled in three courses in the University. Each course represented three different disciplines – Computer Science, Business Management and Product Design. Qualitative and Quantitative data pertaining to the delivery and relevance of the program in relationship to the individual course objectives were collected and synthesized.

Research Context

The Interdisciplinary Service Innovation Program (ISIP) was a first-time endeavor participated by three courses in the university — CMSC 128 Software Engineering in BS Computer Science, MGT 173 Marketing Management in BS Management and VC 26 Design Workshop in the BS Product Design program. The goal of the program was to allow the students to experience creating a project that will improve the services of their target client. During their experience, they applied the core skills taught in their respective programs. They also learned new skills from team members and learned more of the real-world challenges in project development especially that they worked with students coming from different backgrounds.

For Computer Science, ISIP also served as an avenue to teach practical lessons to students on collaboration and communication as these are essential skills in the agile world of Software Engineering and to emphasize best software engineering practices. That is, on top of the application of Software Engineering methods and skills simultaneously discussed in class.

The faculty of the three courses (also referred to in this paper as proponents) followed their respective course syllabi. However, the related course milestones are aligned in a synergy of course requirements. In essence, all individual course requirements were targeted at the development of a service innovation project for the clients. Table 1 summarizes this alignment. This alignment showed that the three courses complement each other in terms of competencies necessary for project development. More specifically, project management and service marketing skills from Management students, a user-based design from Product Design students and problem-solving and software development skills from Computer Science students.

Table 1. Aligned Course Outline for CMSC128, MGT173, and VC26

WEEK	MANAGEMENT	DESIGN	COMPUTER SCIENCE
1	What is service thinking?	Kick-off meeting	
2	Services Marketing	Introduction to Design Thinking	Software Engineering Process Agile Process Models
3	Value proposition	Observation	Requirements Gathering
4	Value proposition	Observation	Requirements Modeling
5	Needs and Opportunity Assessment	Ideation; Brainstorming Tools	Requirements Modeling
6	Service Blueprint	Ideation	Scrum: Project Initiation and Release Planning
7	Concept Evaluation Technique / Positioning	Project Proposal Presentation	Scrum: Sprints Project Proposal Presentation
8	Budgeting	Rapid prototyping	Scrum: Sprint Review and Retrospect, Change Management
9	Testing and Forecasting / Measurement and Control	User Feedback & Iteration; Group consultation	Product Development
10		Iteration; Group consultation	Product Development
11	Group consultation	Iteration; Group consultation	Product Development
12	Group consultation	User Feedback & Iteration; Group consultation	Product Development
13	Group consultation	Iteration; group consultation	Product Development
14	Group consultation	Implementation; group consultation	Product Development
15	Group consultation	Implementation; group consultation	Product Development
16	Submission of Second Draft	Submission of Second Draft	Project Presentation and Deployment
17	Presentation of first 5 groups		
18	Presentation of remaining 5 groups		

The proponents created ten interdisciplinary teams for the program. Each team comprised of 10 to 11 students — four Management, five Computer Science and one to two Product Design. At the end of the program, each team created a service innovation product in the form of a software application. The students achieved this by carrying out their respective course requirements and activities together with other interdisciplinary team members.

The success of the program relied on excellent communication, especially amongst team members. To expedite team formation, the students attended a 3-day team-building activity administered by a third party. The activity was aimed at improving the students' confidence, establish rapport with the other students, and ultimately build teams.

Interdisciplinary, collaborative framework

Figure 1 presents the framework of the Interdisciplinary Service Innovation Program. It is within this framework that an interdisciplinary approach to teaching and learning Software Engineering is induced.

The framework highlights real-world and interdisciplinary collaboration among students from different fields and between teams and real clients. In addition, other structures are placed to support the implementation of the program. The succeeding sections discuss in detail the descriptions of these elements of the framework.

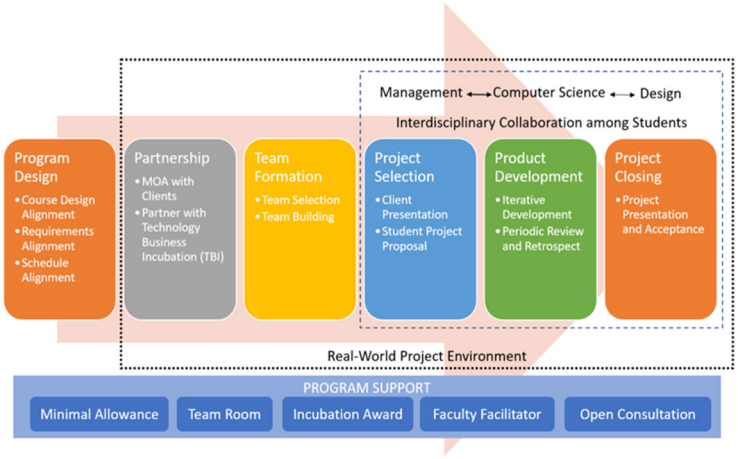

Figure 1. Interdisciplinary Service Innovation Program Framework

Project selection

The idea of an interdisciplinary program primarily sprung from the University's vision to take the leadership role in ICT-driven innovation in Central Visayas, where the university belongs; and secondly, its mission of relevance to the region. Since the services and tourism sectors comprise more than 50% of the region's gross domestic product (Silva, 2017), the proponents from the School of Management and Department of Computer Science believed that improving essential services of the province will provide support to these sectors.

Aside from the traditional approach of lectures and textbooks, the proponents considered using real clients to expose students to real-world technical and management issues when proposing innovations. The public ports and terminals in Cebu City were invited to become the clients for service innovation. The support and active participation of the industry partners are essential to the success of the collaboration. A Memorandum of Agreement formalizing the partnership between the University and Cebu Ports Authority (CPA) and Cebu South Bus Terminal (CSBT) were forged, respectively.

As an overview, the CPA and CSBT representatives presented their organizations, their respective vision and mission, problems and difficulties encountered and possibilities for service innovations to the students. Based on research, available data from the public ports and terminals, customer surveys, and end-user interviews, the teams were tasked to identify opportunities where they can implement technological innovations to improve the public service. Five teams worked on the CPA projects while the other five handled CSBT. The students presented the proposed plans to both the clients and faculty facilitators for approval and recommendations.

To encourage the identification of good projects, the program partnered with the Cebu TTBDO-TBI, the technology incubation hub of the University. With this partnership, the Cebu TTBDO-TBI promised to incubate projects with market potential for further development.

Product development and closing

At the program kickoff, the proponents provided the rationale, learning objectives, and expected outputs of the interdisciplinary program. The Software Engineering faculty provided the students an introduction to the Scrum software development process as well. The proponents acknowledged early on several uncertainties in the project that can be attributed to the teams' inexperience in software development. In the first place, this is the first time that the students are introduced to software engineering. The Scrum software development process was deemed appropriate because it is iterative, and its retrospective nature allows for teams to improve both the product and their dynamics throughout the program.

After the initial presentation of the industry partners, the individual teams investigated further the opportunities for service innovation. The Management team members spearheaded the identification of problems and opportunities. On the other hand, the Computer Science students introduced alternative software solutions, and the Product Design students helped envision product usability. The teams improved on the best software idea. Afterward, they presented the project and product vision to the client and faculty for approval and feedback.

Following the Scrum methodology as a guide, components of the software are iteratively developed during each Scrum sprint. The program had four two-week sprints with new features and functionalities introduced in each sprint. The teams met at the beginning of each sprint to plan for the sprint deliverables. The Management team members were in-charged of gathering information at the client site, such as interviewing frontline employees, support staff, and most especially the passengers or users of the public terminals. User requirements were extracted from these interviews, modeled and implemented in code by the Computer Science students. The design of user interfaces was handled by the Product Design students.

At the end of each sprint, the teams met with the faculty facilitators and their clients. This phase is also known as Sprint Review. During sprint review, the software products were tested for usability and acceptance. Recommendations for improvement and additional requirements were also raised during the meeting so that the teams can accommodate them in the next sprint. The faculty also assessed the performance of the teams and their individual members during Sprint Review.

The Computer Science faculty facilitator, who acts as the Scrum Master, regularly initiated a retrospective meeting, also known as Sprint Retrospect. This meeting was set to discuss what the team needs to improve on in order to increase productivity. It is also during these sessions that the members of the team raised their concerns that need resolving with the help of the Scrum Master. For Software Engineering class, the Sprint Retrospect served as the time when the faculty was able to emphasize software engineering practices that have been discussed in class.

The students were encouraged to meet as often as possible (preferably every day) to discuss their progress on the tasks that they committed to work on. The proponents worked closely with their respective students on the projects.

At the end of the four sprints, each team presented their software products during a final presentation with the clients and other stakeholders.

Colocation is a critical factor in the formation of the Scrum teams. This is especially because the outputs of the Software Engineering students need the active participation of their counterparts from the Management and Design courses. To implement Scrum's recommendation for colocation, the University provided workspaces for all the teams to be able to meet face to face and work on the projects outside of their regular class schedules. The workspaces had large common tables and secure internet connections. Furthermore, each team was allotted minimal allowance to defray travel expenses going to client sites and costs for supplies and materials.

Data Collection and Analysis

The Sprint Retrospect initiated by the Scrum Master provided first-hand data about the experiences of the students. Aside from Sprint Retrospect, an end-of-program evaluation was conducted to gather more data about student interactions, program assimilation, and many others. Each type of evaluation also triangulated pertinent data for analysis. The following evaluations were conducted:

a.) Peer Evaluation

Each team evaluated their team members based on each team member's performance and contribution. Each team member was given a score of one to ten, ten being the highest, in ten performance dimensions.

b.) Program Assessment

The program was assessed by the student participants. The assessment included the relevance of the project to the respective courses and the achievement of the learning outcomes.

There were two types of assessment. The first was a scoring assessment of the general program learning outcomes using a Likert-type scale of 1 to 5. The second was an open-ended question asking the students to detail the highlights (best and least) of their experience.

In addition, a separate program assessment was conducted to target the program's relevance to Software Engineering education for Computer Science students. In this assessment, the students were asked to a) rate the contribution of ISIP to their Software Engineering knowledge and skills and b) detail the specific knowledge/insight or skills that they acquire during the experience.

c.) Focused Group Discussion

Selected teams were interviewed to confirm and discuss the factors that have contributed to their success or failure. During this meeting, the facilitators probed to uncover team dynamics or program structures that influenced the success (or failure) of teams. Two teams, Trak: Bus Tracking System and Bus Fee Payment and Monitoring, were invited to this meeting. The invited teams represented the accomplished group and the needs-improvement teams, respectively.

RESULTS

The list of projects completed by the ten teams is presented in Table 2. Each software product exhibited core functionalities that addressed the immediate concerns related to the clients' services.

Table 2. Software Outputs

Cebu Ports Authority	Cebu South Bus Terminal
Angkla: Berthing Management System	SBT Directory and Travel Assistance
CPA Directory and Passenger Assistance	TRAK: Bus Tracking System
CPA E-ttendance Log and Monitoring	Loading Bay Monitoring
Export Management System	Baggage Loading System
Porter Assistance System	Bus Fee Payment and Monitoring System

Of the ten proposed projects, only two projects did not meet user acceptance due to the poor quality of software presented — Export Management System and Bus Fee Payment and Monitoring System. On the other hand, the Cebu TTBDO-TBI identified the following projects for incubation and client-use: CPA Directory and Passenger Assistance, TRAK: Bus Tracking System, Loading Bay Monitoring, and Baggage Loading System. The clients accepted the other remaining projects, but with revisions.

The clients' acceptance of the majority of these projects signified success of the program in terms of the methods employed in software development.

PROGRAM ASSESSMENT RESULTS

Figure 2 summarizes the students' assessment of the program. Out of 105 students, 79% of the combined students (83 out of 105 students) submitted their evaluation. Results show that the Computer Science students were able to assimilate reasonably well (average of 4.4 out of 5.0) with the target outcomes set by the program, except in the case of leadership (4.0). The relatively low evaluation in "leadership" (compared to the Management students) may be due to the specific roles that each course was assigned to. For instance, the Computer Science and Design students were taking

cues from the Management students as to the direction of the project since the Management students were assigned to identify the needs and opportunities of the target clients.

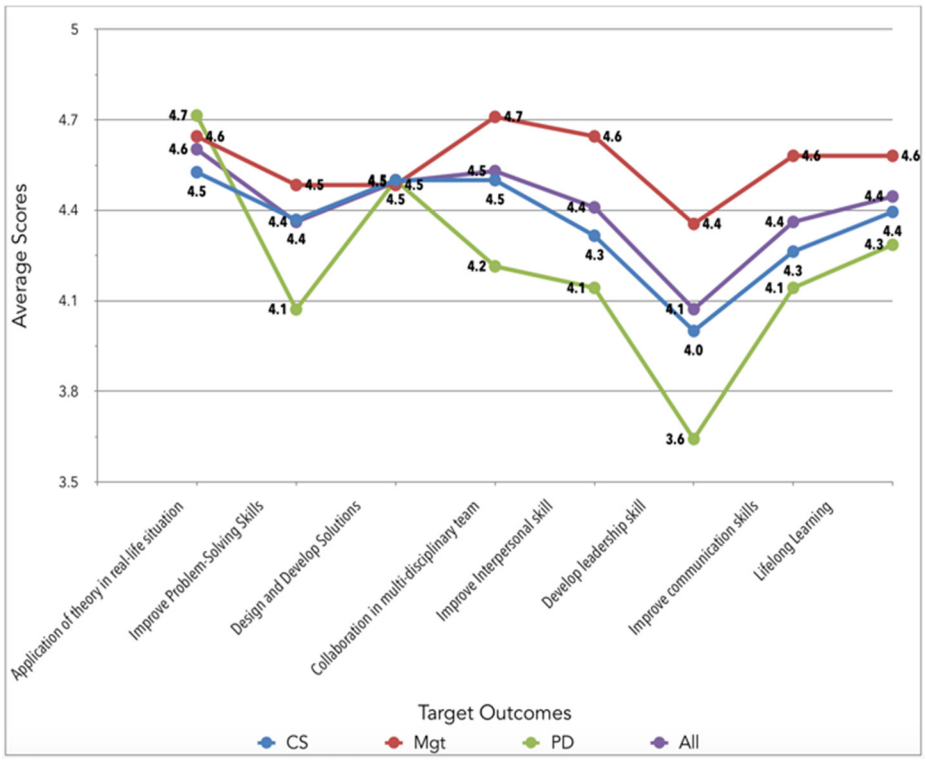

Figure 2. Program Evaluation in relation to the learning outcomes

Notes: CS - Computer Science students; MGT - Management students; PD - Production Design students

The students were also asked to identify highlights of the program that have both positive and negative impact on their learning and in their project. Their responses were coded and from which common themes were extracted. Table 3 and Table 4 enumerate the various themes drawn. Note that because the survey is an open-ended question, respondents can have zero to multiple answers or that respondents can have similar answers. The percentage indicated in the tables that follow represent the counts of similar responses (themes) normalized by the total number of responses. In general, the student responses were consistent with the results of the course evaluation.

It is no surprise that the best thing about the combined course is the Experiential Learning (Active Learning, Collaboration, and Real-World Insights). Working with personalities from different fields, creating service innovation to solve real problems, and dealing with consultants (faculty) and real clients is a far cry from just learning from textbooks. In addition, working on a project relevant to the society provided a different meaning to the project they are working on. Based on the evaluation, these factors have made a positive impact on the learning of the students under the program. It is likely that this positive outlook towards the program must have contributed to the success of the majority of the teams. Moreover, the students perceived other program elements such as Product Development Methods, Team Building Activities, and Financial Assistance as motivating factors.

Table 3. Program Highlights (Best)

	DESCRIPTION	% of Responses	EXAMPLE ENTRIES
Active Learning	Learning new technical skills, experiential learning	21.1%	"I learned a lot from it, not only what was being taught but also from our own experience."
Collaboration	Being able to work with others	21.1%	"Collaborating with students from different courses exposed me to the possible challenges that I will be facing when I graduate and start working."
Social Skills Improvement	Improving communications skills	13.7%	"Helped me improve as a person involving me in social discussions"
Real-world Insights	Real client, real project, and real challenges	12.4%	"Real life problems were given and we had to work as if we were a start-up team. Experience-wise, it was really good."
Project Relevance	Product's usability	8.7%	"Being able to develop something, specifically an application/project that aims for the betterment of another system."
Working in Teams	Experiencing team work; being part of a team	8.1%	"The sense of struggling together and winning together."
Product Development Activities	Processes involved in product development	7.5%	"Group meetings. Delivery of the products (sprints)"
Integration of Course	Synchronization of individual course requirements	3.1%	"…communicating and translating expectation from one field to another."
Team Building	Third-party team building activity	3.1%	"The event/things I like best is being able to experience the Kool Adventure Camp"
Financial Assistance	Allotment of financial assistance	1.2%	"the course provided me the allowance we need to complete the project."

Total No. of Coded Responses = 161

Table 4. Program Highlights (Least)

	DESCRIPTION	% of Responses	EXAMPLE ENTRIES
Time	Project development requires more time	26.0%	"the thing I like the least is how much time is dedicated to this course."
Schedule Mismatch	Course or student schedules do not match	24.0%	"We have different free time which makes it hard for us to meet together."
Commitment to Task	Members commitment to team agreements	6.3%	"Other members who would cram which affects the whole group/system."
Pressure	Stress from pressure	6.3%	"Sleepless NIGHTS because of deadlines."
Team Formation	How the teams are formed	6.3%	"nothing, just my groupmates"
Team Storming	Problems among team members	6.1%	"It takes most of the time, sometimes the misunderstanding with the groupmates"
Prerequisite	Lack of prerequisite knowledge or skills	5.2%	"There could also be more talks and class that taught skills necessary for the project such as CSS for the CMSC and UI and UX design for Product Design."
Communication Problems	Inaccessibility of members	4.2%	"It's also hard working with other courses because people have different say on things."
Faculty Coordination	Unclear instructions from faculty	4.2%	"Profs from the different program have a different vision for the project"
Project backlogs		4.1%	"What I really don't like are the frustrations that are __ when the software's bugs seemed to be so hard to fix"
Changes in schedules	Unprecedented schedule changes	2.1%	"What I didn't like about the course is the abrupt change of schedules."
Overlapping roles	Project roles tend to overlap creating confusion	2.1%	"The multi-disciplinary set up has set certain tasks on each but some tends to get outside/beyond the boundary of assigned tasks and makes the distribution of tasks unequal"
Process Adaptation	Slow adaptation to product development process	1.0%	"Working in small increments is a new concept for me, and I've never fully adapted to it."
Expensive	Has financial implications	1.0%	"very demanding of the time, energy and money (but it's okay)"
Other Course Requirements	The tendency to focus on individual course requirements	1.0%	"I believe that the management students were not of much help to the team considering that we were more concerned about finishing our final paper than developing the system in general "

Total No. of Coded Responses = 96

Since it was ISIP's pilot run, it was also essential to recognize the factors that have had adverse effects on the performance of the students. These factors need to be considered for the program's future improvement. Table 4 shows that Time is the most significant factor that impeded project development. The students had difficulty finding a common time to meet because the schedule of the courses was different for each class. As a result, most of the teams who did well needed to allocate more time (i.e., working at night and on weekends) to finish the projects. On the other hand, the most common complaints of the teams that didn't do well in the program were the perceived Lack of Commitment of their teammates to the assigned tasks and Communication problems. These too can be a consequence of schedule constraints.

The proponents recognized from the start that one of the challenges in this pilot run will be the Formation of Teams. Since the students come from different courses, with different personalities and priorities, the faculty contracted an external organization to conduct team-building activities with the students and fast-track the process of Team Formation. While a number of students appreciated this activity (Table 3, Team Building), there are those who did not perceive the activity as effective because the teams during the team-building activity were not grouped according to the actual teams. As one student noted:

> *"The Kool Adventure Camp should and must be grouped according to the actual teams. I have been to KAC 7 times already and it never failed to unite the most diverse group."*

PEER EVALUATION RESULTS

The Peer Evaluation instrument measured how well each team member performed within the team in ten performance dimensions. The average of students' mean of scores, 8.9, generally depicts a substantial cooperation among team members in each team. Table 5 summarizes the peer evaluation results of the teams in each performance dimension.

Table 5. Peer Evaluation Summary

	Performance Dimensions									
Teams	Attendance at Meeting	Punctuality	Contributes Ideas	Initiates Ideas	Accepts Responsibilities	On-time Submission	Positive Work Attitude	Organized	Preparedness	Knowledgeable
SBT Directory and Passenger Assistance	**9.13**	9.17	9.36	9.36	*9.69*	*9.69*	9.63	9.50	9.56	9.32
TRAK: Bus Tracking System	**9.27**	9.57	9.64	9.43	9.92	9.55	*9.85*	9.67	9.52	9.52
Loading Bay Monitoring	**8.98**	9.24	9.35	9.22	9.53	9.40	9.26	9.42	9.35	*9.58*
Baggage Loading System	**8.39**	8.81	9.42	9.15	*9.58*	9.24	9.56	9.32	9.18	9.35
Bus Fee Payment and Monitoring System	8.52	8.81	**8.50**	8.38	8.96	8.59	*9.13*	8.64	8.73	8.88
Angkla: Berthing Management System	**8.00**	8.08	8.55	8.28	*8.83*	**8.00**	8.52	8.56	8.35	8.57
CPA Directory and Passenger Assistance	**8.66**	8.78	9.03	9.03	9.24	8.84	*9.38*	9.32	9.06	9.00
CPA E-ttendance Log and Monitoring	8.91	8.65	**8.54**	8.57	*9.19*	8.68	8.91	8.97	8.80	8.99
Export Management System	8.57	8.63	8.29	8.26	*8.86*	**8.17**	8.63	8.48	8.44	8.67
Porter Assistance System	**8.31**	8.61	8.61	8.58	8.94	8.81	*9.30*	8.98	8.68	9.12

The minimum and maximum team averages are highlighted in the table. The lowest average ratings are written in bold, while the maximum average ratings are underlined. It is interesting to note that the lowest average performances are in the Attendance to Meeting, Idea Contribution, Punctuality, and On-time Submission dimensions. These low-score dimensions coincide with the negative factors presented in Table 4. That is, all these are related to Time. The highest scores are in the following aspects: Acceptance of Responsibilities, Positive Work Attitude, and Knowledge about the project.

The low evaluations for the Attendance, Punctuality and On-time Submission can be attributed to the differing schedules of the three courses. One of the most common complaints of the students across all classes was finding a common time for all the group mates to be able to meet face to face. To recover lost time due to the lack of common schedule, the groups met online and in smaller groups to discuss their projects.

The high ratings for Acceptance of Responsibilities and Positive Work Attitude of the students show the favor of the students to the multidisciplinary class.

Software Engineering Evaluation Results

This section presents the results of a separate survey to assess the program specific to Software Engineering education. This survey was participated in by the Computer Science students.

The results show that out of the 87% respondents (47 out of 54 Computer Science students), 72% started the program with a Poor-Fair skill or knowledge self-rating. At the completion of the course, 62% rated their perceived level of skills or knowledge Very Good to Excellent, while 32% rated having a Satisfactory skill or knowledge. This means that the students have generally learned and acquired new skills during the program. On top of that, 70% (Very Good to Excellent) attributed this learning to the program itself. Table 6 refers to the counts of students' program evaluation responses.

Table 6. CMSC 128 Program Evaluation

	Poor	Fair	Satisfactory	Very Good	Excellent
Level of skill/knowledge at the start of the course	11	23	10	1	2
Level of skill/knowledge at the end of the course	0	3	15	22	7
Level of skill/knowledge required to complete the course	1	3	12	21	10
The contribution of course to skill/knowledge	0	3	11	20	13

The students are further asked about the specific knowledge or insights and skills that they acquired during the program. The probe was narrowed to the three major phases of software development — (a) Requirements Gathering, (b) Design and Development of Software and (c) Software Testing. Just as in the previous surveys, the raw, open responses are coded, categorized according to skills or insights, and tabulated. The succeeding sections present these tabulated results. The percentage indicated in Tables 7 to 12 represent the counts of similar responses (themes) normalized by the total number of responses.

Table 7. Skills acquired during Data Gathering

Skills	% Responses
Asking the Right Questions	33.3%
Team Communication, Communication, Involve Client through Communication	25.9%
Active Listening	7.4%
Negotiation, Negotiating conflicting requirements	7.4%
Problem Solving, Systems-Thinking	7.4%
Requirements Analysis	7.4%
Patience	3.7%
Requirements Modeling	3.7%

Total no. of responses coded as Skills = 27

Since the collaborative program is the first time that the Computer Science students are introduced to software development, it is just likely that the top skill that they acquired during requirements gathering was to ask the clients right questions. Table 7 tabulates the students' responses on the skills acquired by the students during Requirements Gathering.

The results also show that Communication skills, a skill essential in team-based projects, emerged as one of the prominent skills that the students acquired from the program. Equally important is the articulation of the technical skills for software development — Problem Solving, Requirements Analysis and Modeling.

On the other hand, Table 8 summarizes the insights that students have gained from their experiences during project development. Apparently, the tenets of agile development on collaboration appeared to be the most learned from their experience. It is supposed that the iterative and retrospective nature of Scrum provided the students the avenue to correct their mistakes of the previous sprint. The mistakes were resolved through collaboration with the client and their team members. It is also noted that the use of User Stories and Story Maps as modeling tools have helped the student in their project. Although minimal, it was noted that some students articulated the idea of 'Shift in team roles as the need arises'. This insight is one that is descriptive of an agile team. However, in Table 4 Program Highlights (Least), this 'shift in roles' is seen as an overlap in functions and that this had a negative effect on the team. It appears now that some members, apart from the Computer Science members, have not yet fully assimilated this characteristic of agile software development.

Table 8. Data Gathering Insights

Category	Insights/Knowledge	% Responses
Data Gathering	Client requirements maps User Satisfaction	2.2%
	Consider all stakeholder in requirements gathering	8.7%
	Progressively elaborated nature of requirements	8.7%
	Negotiating conflicting user requirements, establishing win-win solutions	4.3%
	Meaningful client involvement in establishing requirements	21.7%
	Observation as a requirements gathering tool	2.2%
	Requirements specification and verification	4.3%
	Take heed of details	2.2%
	The team should understand requirements.	2.2%
Modeling	Appreciate user-focus	2.2%
	Incorporating insights from other people in modeling the system	2.2%
	Modeling is important to meet highest user experience	2.2%
	Prototyping to model system	2.2%
	Use of Story Maps to give perspective on how software should behave	2.2%
	User Stories to model requirements	8.7%
Planning	Importance of planning	6.5%
	Planning takes more time than coding	2.2%
	Prioritizing requirements helps in planning work	2.2%
	Shift in team roles as the need arises	2.2%
Others	Make software work can be prioritized more than data gathering and modeling because of time constraints	2.2%
	Importance of Team communication	6.5%
	Use of technology to communicate is more convenient	2.2%

Total no. of responses coded as Insights = 46

Moreover, Tables 9 and 10 present the summary of the skills and insights that the students acquired concerning Design and Development. The technical skills acquired by the students are commendable considering that, for some teams, the technology stack used for their project was relatively new. More than anything the students learned resourcefulness as a team in order to deliver their product (refer to Table 9) Self-learning new technologies.

It is also significant that the students learned the relevance of good requirements gathering to software design (refer to Table 10) Communication remains to be an essential element in the development of projects as experienced by the students. In addition, the students appreciated the existence of a Software Engineering methodology to guide them in their project development. Although minimal, the students' insight on leveraging team members' expertise was highlighted. This insight is practically the essence of the interdisciplinary program.

Table 9. Skills acquired during Design and Development

Skills	% Responses
Self-learning new technologies	29.4%
Teamwork	29.4%
Database and software development	5.9%
HTML	5.9%
Mobile development	5.9%
Pair programming	5.9%
Practice UI design	5.9%
Prototyping	5.9%
User interface design	5.9%

Total no. of responses coded as Skills= 17

Table 10. Design and Development Insights

Category	Insights/Knowledge	% Responses
Design	Client perspective and developer perspective are different	2.3%
	Consistency in user-interface design	2.3%
	Design of software should complement client environment, user-focused	6.8%
	Importance of understanding requirements for a better design, user interface design, usability	15.9%
	Long-term thinking	2.3%
	Not everyone knows user interface design, Product Design team members provided good inputs for user interface design	4.5%
	Prototyping as design verification	2.3%
	Use of story maps in the design	2.3%
	User interface design is just as important	2.3%
Development	Commitment to assigned tasks, timely submission of deliverables	4.5%
	Communication is key to successful teams — between dev and design team members and between team and client	9.1%
	Experienced how agile methods work in rapid software development, Fail Fast	4.5%
	Importance of task scheduling	2.3%
	Programming is difficult when scope and limitations are not clear	2.3%
	Since tasks are divided among members, team members agree on specs or versions for better integration of outputs	4.5%
	Software development process helped, saves time, make dev work easier	9.1%
	Sprints	4.5%
	The more meetings, the better the sprint deliveries	2.3%
	Training technology stack before implementation of design will improve delivery schedule	2.3%
	Utilization of Designs patterns	2.3%
Others	Designing and developing software are two different things	2.3%
	Designing before programming helped	2.3%
	Division of work	2.3%
	Importance of Time management	2.3%
	Use the expertise of each team member in the design and development of software	2.3%

Total no. of responses coded as Insights = 44

And finally, this section presents the results for Software Testing. In terms of testing skills, the results (Table 11) show that the students have not fully implemented the testing skills required of them, i.e., writing test cases, code review, etc. This can be attributed to the time required to introduce in-depth lecture on testing and monitor application of proper testing techniques during sprints. A five-month (one-semester) Software Engineering course is undoubtedly too short.

Table 11. Software Testing Skills

Skills	%Responses
Handle different types of users	25.0%
Hard work	12.5%
Meticulous, pay attention to details	12.5%
Open to suggestions	25.0%
Patience	12.5%
Understanding and foresight in order to be thorough	12.5%

Total no. of responses coded as Skills = 8

Table 12. Software Testing Insights

Category	Insights/Knowledge	% Responses
Value of Testing	Importance of testing, site testing with end-users	18.4%
Test Process	Case coverage during testing should ensure usability	2.6%
	Different users to test the system affords different perspectives to improve the system	2.6%
	For something to be perfect, it has to be tested multiple times, rigorous testing before release	5.3%
	Integration test the most difficult part of testing	2.6%
	Proper allocation of time for testing	2.6%
	Site testing is better	5.3%
	Test cases done first, use of test cases	5.3%
	Testing and validation takes more effort than previously thought, more than what's in the books	5.3%
	Testing at different levels (unit, integration, system) that tackle different stages of development	2.6%
	Testing at the point of view of the user	2.6%
	Testing of increments works	2.6%
	Use of unguided user testing	2.6%
Recommendation	More on automated testing	2.6%
	More time on testing	2.6%
	More training on TDD	5.3%
Quality Mindset	Always code with the user in mind	2.6%
	Learning from previous sprints helped improve software	2.6%
	As programmers, we can easily lose sight of clients' perspective	2.6%
	Communication improves design	2.6%
	Different team perspectives help in design and understanding of systems	2.6%
	Getting the right requirements saves having to do rework	2.6%
	The quality mindset in product development	2.6%
	Understanding the product in order to foresee issues and test the product better	2.6%
Others	Murphy's Law	2.6%
	Practice marketing the product	2.6%
	Requirements are also known during testing and validation	2.6%

Total no. of responses coded as Insights = 38

The importance of software testing though was emphasized by the students as one of the knowledge gained through their experience. The majority of the responses expressed the value of testing to software quality. Interestingly, Table 12 shows that students also learned from their experience the impact of other Software Engineering elements such as communication, user-focus, retrospection, and proper requirements gathering to the quality of the software they produced.

It was also articulated in the responses that the students clamor for an in-depth discussion of advanced testing techniques such as automated testing. This is because lecture classes ran out of time for discussing these topics.

DISCUSSION

This case study explored the enhancement of Software Engineering education through an interdisciplinary approach. The practiced pedagogy was improved by introducing a different environment where interdisciplinary students, faculty facilitators, and real-world clients collaborate to learn from each other's disciplines. This environment is seen as a microcosm of the industry's diverse Software Engineering teams and where both technical and soft skills can be learned.

The three courses in the University — Software Engineering, Marketing Management, and Design Workshop — prove to be complementing courses as they enhance each other's learning outcomes as applied to real-world software projects. More specifically, each of the courses' learning outcomes corresponds to the skills and experience needed in software development.

In this discussion, the results presented in the previous section are synthesized in three frames – (1) students reception of the interdisciplinary, collaborative project development, (2) factors that influence project completion and (3) software engineering skills and knowledge acquired during the interdisciplinary program.

PROGRAM RECEPTION

The program evaluation results show that the ISIP framework was widely accepted by the students. This acceptance is primarily due to active learning. Students are inclined to the program because the approach is in line with their learning style -- visual, sensing, inductive, and active (Felder & Silverman, 1988). This characteristic of the program is relevant because, according to literature, the retention of knowledge and skills improve when students are actively involved in the process of learning (Razmov, 2007).

In the context of interdisciplinarity, the results show that the collaborative nature of the program is also well received by the students (Figure 2 and Table 3). It should be noted that this study implemented a collaboration among students in multiple disciplines. The argument of collaboration, therefore, is taken in the light of interdisciplinary collaboration. In addition, the novelty of the course integration (Table 3) seemed to excite the students undertaking the program.

Furthermore, the acceptance of a majority (8 out of 10) of the innovation projects by the clients and incubation of some is an evidence of successful implementation of Scrum software development practices.

FACTORS THAT INFLUENCE TEAMS AND PROJECT SUCCESS

There are several challenges to incorporating interdisciplinary approach in a course design. Newell (1994) presented several considerations for designing interdisciplinary courses such as the composition or assembly of interdisciplinary faculty teams, topics that influence creative tension between disciplines, student interests, and other mundane considerations such course requirements, scoring mechanisms, and course credits.

For this case study, collaboration-related challenges were also considered since this study, in essence, followed a collaborative approach. Bower and Richards (2006) shared several reasons why it is a challenge to implement a collaborative approach in computer science education. They offered the following reasons. (1) Student reticence due to the apprehension of not getting the appropriate credit from collaborative work. Also, Computer Science students may want to avoid peer interactions influenced by self-confidence, lack of enjoyment or perceived cost in terms of effort in collaborating with others. (2) Faculty can be discouraged by the effort that needs to be spent in creating a collaborative environment or creating collaborative activities. (3) If students work in teams and produce a combined deliverable, it can be difficult to accurately assess the contribution of each student and fairly apportion marks. (4) Academics may avoid collaboration because they will be uncomfortable with the technology that is needed to support collaboration. This line of reasoning is resonated with the students as well. Students fear that they will not be given ample support during their collaborative work. And finally, (5) students fear the notion of other team members that are simply free-loading.

Most of the concerns mentioned were considered before and during program implementation. However, this study continued to investigate several other factors that can influence the collaborative, interdisciplinary curricular intervention. In the first place, it is this study's goal to enhance student learning. Both positive and negative factors are identified based on survey results, group discussion and collective observation of the faculty involved.

This section answers this study's first research question – What factors strongly influence a successful software project in an interdisciplinary project environment?

Positive factors

a. Project-Course Relevance
   This factor refers to the overall design of the program in relation to course objectives. The results show that students responded positively to the activity because they were able to experience what had been taught in the classroom. Learning, therefore, was active. This activity actually matched the innate learning styles of computer science students according to Felder and Silverman (1988).

b. Product-Community Relevance
   Creating software solutions that address real-world problems of real-world clients increased the students' motivation to complete the project. This motivator is also present in most case studies cited that had employed collaborative approaches with real clients (Chase, et al., 2007; Chen & Chong, 2011; Flener, 2006).

c. Team cohesion
   Team cohesion refers to the degree of collaboration among interdisciplinary members of the team. The students generally responded positively to the presence of teamwork even when the team is struggling to accomplish some tasks or that even when the members spent more time on their projects than their other courses. It appeared that in this case, Bower and Richards' (2006) student reticence factor is overcome by team identity or teamwork. Peer evaluation results suggest that positive work attitude and acceptance of responsibilities are good characteristics that must be embodied by each team member to foster teams.

d. Financial Assistance
   This support was appreciated because real-world practice required them to spend for on-site visits to clients. The visits were intended for requirements gathering, clarification, and product testing. Financial assistance covered travel, supplies and meals allowance. This support coincides with Bower and Richards' (2006) administrative support factor.

Negative Factors

a. Schedule alignment
The Scrum software development methodology suggests for the members of the team to be collocated in order to speed up communication, hence speeding-up development. This aspect was very challenging because the team members are primarily students coming from different programs. Despite the allocation of team rooms, students still had difficulties meeting because of the schedule conflict. The students perceived the lack of common schedule as the number one roadblock to their success. Because student schedules did not match, many of the problems such as absence to meetings, communication, and submission of task outputs emerged.

It is therefore recommended, as an administrative support factor (Bower & Richards, 2006), to align student schedules so that they can find time to collocate together as a team.

b. Project Vision to be clearly translated into individual course requirements
Although the program's ultimate output is the software project, the individual courses have specific course requirements that the student needed to comply. It is noted that this separate submission drove some students to lose sight of the overall objective and focus on their own course deliverables instead. While the supposed learning outcomes were related, the students still had the tendency to prioritize their individual course requirements.

Apparently, the Scrum discussion at project kickoff was not enough to relate individual course requirements to software development activities. The contribution of the respective course requirements to the overall intended project output needs to be further emphasized.

c. Faculty Coordination
Students complained that it is confusing when faculty facilitators convey differing, worst conflicting, instructions or perspectives. These instructions include the schedule for submissions and interpretation of software requirements. Newell (1994) emphasized this challenge when calling in colleagues from other teams for collaborative assistance. He mentioned that the faculty involved in interdisciplinary courses should "hold an uncomfortable idea to the light, turn it around, see how it might relate to more familiar ideas" (Newell, 1994). This case taught us, therefore, that, more than the students, the faculty should also practice interdisciplinary thinking. More faculty coordination is recommended.

d. Team Building and Retention
Even with the 3-day team-building activity, the students noted in their evaluation and observations that there is a need to strengthen the teams continuously. Some teams had a rough start. There were those that degraded during the team's norming stage.

It is recommended that the teams are grouped together during the three-day team-building activity and that everyone should be required to attend. Moreover, short daily meetings (Stand-up Meetings) should be required to facilitate communication among team members. Again, schedules must be aligned.

e. Overlapping Roles
In an agile environment, overlapping roles should be seen as a result of review and retrospection to speed up product development and creative tension (Newell, 1994). In this particular case study though, overlapping roles became a deterrent to some Computer Science and Product Design students. Apparently, as each is considered "experienced" in their fields, efforts to correct each other had a destructive tendency. Newell (1994) warned against this kind of behavior because the interdisciplinary contribution of the activity is lost. The creative tension is lost if the disciplines are seen as specializing in different parts of a whole, such as in the case of students holding on to

their roles based on their expertise. This conflict can be resolved with appropriate intervention from faculty.

f. Course and Program Duration
It was observed and expressed in the survey that the program duration is too short in relation to its general objectives. The program runs for five months only, taken simultaneously with Software Engineering lectures and other courses. It is recommended to extend the course and program to two semesters instead. This will give students ample time to explore each other's expertise and build their solutions.

KNOWLEDGE AND SKILL ACQUISITION

The second research question focused on the students' perceived knowledge or skills related to Software Engineering that is acquired or enhanced during the program. The collaborative, interdisciplinary framework, in itself, already provided an authentic and realistic experience for students within which practical knowledge of software engineering practices can be acquired. In this discussion, the results on knowledge/skill acquired are synthesized to verify whether the insights or skills acquired are relevant to their future profession as software engineers. To do this, the discussion is framed around Lethbridge's (2000) Knowledge Gaps.

Lethbridge (2000) highlighted three forms of Knowledge Gaps. First, the Educational Knowledge Gap. This knowledge gap refers to the difference between the amount of knowledge learned in education and the particular knowledge's importance in the industry. Second is the On-the-Job Learning Gap. This refers to the knowledge areas that might not have been emphasized in class but learned in the profession instead. And third, the Current Knowledge Gap. This knowledge gap refers to the knowledge areas that are deficient at the time of study but are deemed necessary to software engineering. Because this case study's objective was to reinforce classroom learning through an interdisciplinary approach to project development, the students' responses were summarized according to the last two forms of knowledge gaps.

Table 13 matches the students' responses to the knowledge areas that are typically learned On-the-Job (Lethbridge, 2000). The analysis is focused on Software Engineering Methods and Essential Subsystem Design categories as these are the target knowledge areas for the interdisciplinary program.

Table 13. Attributed Learning after Education

Category	Topics Learned on the Job (or forgotten since education)	Insights/Skills Acquired according to Students' Responses?	Remarks
Software Engineering Methods	Requirements gathering and analyses	✓	
	Analysis and Design methods	✓	
	Testing, Verification and Quality Assurance	✓	
	Software Reliability and Fault Tolerance		Evidenced in software produced
	Maintenance, Reengineering, and Reverse Engineering		Not introduced
Essential Subsystem Design	Human-computer interaction/user interfaces	✓	
	Databases	✓	

It was noted that majority of the perceived knowledge/skills acquired during the program are the same knowledge/skills acquired by some software engineers on-the-job.

Table 14, on the other hand, matches the students' responses to the knowledge areas that are recommended as additional employee training because it may be lacking or missing in the software engi-

neer professional. The top 10 topics with the greatest knowledge gap according to Lethbridge are utilized in the synthesis. Based on Table 14, it is remarkable that the students learned Negotiation and Leadership early on in their studies.

Table 14. Attributed Learning against Education Gap

Rank	Topic	Insights/Skills Acquired according to Students' Responses?	Remarks
1	Negotiation	✓	
2	Human-computer interaction/user interfaces	✓	
3	Leadership	✓	
4	Real-time System Design	✓	
5	Management		Observed in Management team members
6	Software Cost Estimation		Observed in Management team members
7	Software Metrics		
8	Software Reliability and Fault Tolerance		Evidenced in software produced
9	Ethics and Professionalism		Observed in quality of product and presentation
10	Requirements gathering and analyses	✓	

CONCLUSIONS AND RECOMMENDATIONS

This study presented a technique for integrating interdisciplinary approach to Software Engineering Education. Literature review suggests that for a Software Engineering course, a collaborative approach through projects is advantageous (Chase et al., 2007; Chen & Chong, 2011; Flener, 2006; Giraldo et al., 2010; Kirti et al., 2015). This case study advanced the current pedagogy by involving other related disciplines in the project framework. Specifically, the study integrated Business Management and Product Design disciplines. As in previous researches involving collaboration, the findings revealed that students generally appreciated the course intervention citing that the knowledge learned during the program can be used in their future professions. In the context of Software Engineering Education, this case study provides new evidence that the academe can still improve the delivery of the course. This improvement is grounded on the Situated Learning and Learning-by-Doing theories proposed by Lave (1988, as cited in Ellis, Demurjian & Naveda, 2009)).

Consistent with the work of Bowser and Richard (2006), this study proves that academics can easily integrate interdisciplinary learning by applying it in a problem-solving (Nikitina, 2006), collaborative environment. Several factors that can influence the success or failure of the integration are presented in this paper as well.

Many other research related to Software Engineering Education can be made from this case study. It is recommended that the interdisciplinary approach presented in this case be applied and studied further in other universities. Moreover, it is interesting to know and eventually understand the interactions between interdisciplinary team members in the conduct of Software Engineering practices while working on their projects. More specifically, what creative tensions arise and how do the interdisciplinary teams handle the discourse. Ultimately, the proponents recommend a refinement and elaboration of the elements of the interdisciplinary framework presented in this paper towards a proposal of an integrated course module for Software Engineering Education.

ACKNOWLEDGMENTS

Our thanks to Cebu Port Authority and Cebu South Bus Terminal for their support and participation in the program. Also, for the UP Cebu Technology Business Incubator (TBI) for their assistance. The authors would also like to acknowledge the UP Cebu Office of Continuing Education and Pahinungod (OCEP) for their financial support on this program.

REFERENCES

Beard, D. (2014). *Learning from each other: An interdisciplinary approach to Software Engineering* (Unpublished Honours Degree Thesis). Australian National University, Australia.

Blaschke, L. M. (2012). Heutagogy and lifelong learning: A review of heutagogical practice and self-determined learning. *International Review of Research in Open and Distance Learning, 13*(1), 56-71. https://doi.org/10.19173/irrodl.v13i1.1076

Bower, M., & Richards, D. (2006). Collaborative learning: Some possibilities and limitations for students and teachers. In L. Markauskaite, P. Goodyear, & P. Reimann (Eds.), *Proceedings of the 23rd Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education: Who's Learning? Whose Technology?, 1*, 79-89.

Capretz, L. F. (2003). Personality types in software engineering. *International Journal of Human-Computer Studies, 58*(2), 207-214. https://doi.org/10.1016/S1071-5819(02)00137-4

Cooper, M. (n.d.). *One proven way to boost software quality: Increase your QA team's diversity*. TechBeacon. Retrieved from https://techbeacon.com/improve-software-quality-qa-team-diversity

Chase, J. D., Oakes, E., & Ramsey, S. (2007). Using live projects without pain: The development of the small project support center at Radford University. *SIGCSE Bulletin, 39*(1), 469-473. https://doi.org/10.1145/1227504.1227468

Chen, C. & Chong, P. (2011). Software engineering education: A study on conducting collaborative senior project development. *Journal of Systems and Software, 84*(3), 479-491. https://doi.org/10.1016/j.jss.2010.10.042

Ellis, H. J. C., Demurjian, S. A., & Naveda, J. F. (Eds.). (2009). *Software engineering: Effective teaching and learning approaches and practices*. Hershey, PA: IGI Global. https://doi.org/10.4018/978-1-60566-102-5

Felder, R., & Silverman, L. (1988). Learning and teaching styles in engineering education. *Engineering Education, 78*(7), 674-681.

Flener, P. (2006). Realism in project-based software engineering courses: Rewards, risks, and recommendations. In *21st international conference on CIS, ISCIS'06*, 1031-1039. Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/11902140_107

Giabbanelli, P., Reid, A. & Dabbaghian, V. (2012). Interdisciplinary teaching and learning in computing science: three years of experience in the MoCSSy program. *Proceedings of the 17th Western Canadian Conference on Computing Education, WCCCE 2012*. https://doi.org/10.1145/2247569.2247586

Gannod, G. C, Burge, J. E., & Helmick, M. T. (2008). Using the inverted classroom to teach software engineering. *Proceedings of the 30th International Conference on Software Engineering*, 777–786. https://doi.org/10.1145/1368088.1368198

Giraldo, F. D., Collazos, C. A., Ochoa, S. F., Zapata, S., & de Clunie, G. T. (2010). *Teaching software engineering from a collaborative perspective: Some Latin-American experiences*. In 2010 Workshops on Database and Expert Systems Applications, 97-101. https://doi.org/10.1109/DEXA.2010.39

Goldsmith, A., Hamilton, D., Hornsby, K., & Wells, D. (n.d.). Interdisciplinary approaches to teaching. In *Pedagogy in Action: the SERC portal for Educators*. Retrieved from https://serc.carleton.edu/sp/library/interdisciplinary/interdisciplina.html

Gottschalk, P., & Solli-Sather, H. (2007). Knowledge transfer in IT outsourcing relationships: Three international case studies. *International Journal of Innovation and Learning, 4*(2), 103-111. https://doi.org/10.1504/IJIL.2007.011687

Jaccheri, L., & Sindre, G. (2007). Software engineering students meet interdisciplinary project work and art. *Proceedings of Information Visualization, 2007. IV '07. 11th International Conference*, 925-934. https://doi.org/10.1109/IV.2007.102

Kirti, G., Sureka, A., & Varma, V. (2015). A case study on teaching software engineering concepts using a case-based learning environment. *Proceedings from 1st International Workshop on Case Method for Computing Education (CMCE)* in conjunction with the 22nd Asia-Pacific Software Engineering Conference (APSEC 2015), New Delhi, India.

Lethbridge, T. C. (2000). What knowledge is important to a software professional? *Computer, 33*(5), 44-50. https://doi.org/10.1109/2.841783

Liang, T., Liu, C., Lin, T., & Lin, B. (2007). Effect of team diversity on software project performance. *Industrial Management & Data Systems, 107*(5), 636-653. https://doi.org/10.1108/02635570710750408

McCuskey, D., & Conaway, W. (1955). The interdisciplinary approach. *Educational Leadership, 12*(7), 395-401.

Mead, N. R. (2015). Industry/university collaboration in software engineering education: Refreshing and retuning our strategies. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 273-275. https://doi.org/10.1109/ICSE.2015.156

Newell, W. H. (1994). Designing interdisciplinary courses. *New Directions for Teaching and Learning, 58*(58), 35–51. https://doi.org/10.1002/tl.37219945804

Nikitina, S. (2006). Three strategies for interdisciplinary teaching: Contextualizing, conceptualizing, and problem-centering. *Journal of Curriculum Studies, 38,* 251-271. https://doi.org/10.1080/00220270500422632

Oches, R. (2012). *Ethanol & Sustainability teaching: Integrating business, public policy and science* [PowerPoint slides]. Retrieved from https://serc.carleton.edu/integrate/teaching_materials/interdisciplinary_format.html

O'Leary, M., & Azadegan, S. (2005). An interdisciplinary approach to scientific modeling and simulation. *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network*, 372-377. https://doi.org/10.1109/SNPD-SAWN.2005.17

Razmov, V. (2007) Effective pedagogical principles and practices in teaching software engineering through projects. *37th ASEE/IEEE Frontiers in Education Conference*, Milwaukee, WI.

Reichlmayr, T. (2005). Enhancing the student project team experience with blended learning techniques. In *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference, T4F.* https://doi.org/10.1109/FIE.2005.1611982

Rus, I., & Lindvall, M. (2002). Knowledge management in software engineering. In *IEEE Software, 19*(3), 26-38. https://doi.org/10.1109/MS.2002.1003450

Silva, V. (2017, 3 July). Services, Tourism will still be CV's economic drivers if... *Cebu Daily News*. Retrieved 16 January 2018 from http://cebudailynews.inquirer.net/138210/services-tourism-will-still-cvs-economic-drivers

Stein, M. (2002). Using large vs. small group projects in capstone and software engineering courses. *Journal of Computing Sciences in Colleges,* 17(4), 1-6.

Stettina, C. J., Zhou, Z., Bäck, T., & Katzy, B. (2013). Academic education of software engineering practices: Towards planning and improving capstone courses based upon intensive coaching and team routines. *26th International Conference on Software Engineering Education and Training (CSEE&T)*, 169-178. https://doi.org/10.1109/CSEET.2013.6595248

The University of Sydney School of Education and Social Work. (2018). *Collaborative learning*. Retrieved from http://sydney.edu.au/education_social_work/learning_teaching/ict/theory/collaborative_learning.shtml

Williams, K. Y., & O'Reilly, C. A. (1998). Demography and diversity in organizations: A review of 40 years of research. *Research in Organizational Behavior, 20*, 77-140.

Wong, W. B. L., McGuire, M., & McDonald, J. (1996). An interdisciplinary approach to multimedia systems education: the Otago experience. *Proceedings 1996 International Conference Software Engineering: Education and Practice*, 394-399. https://doi.org/10.1109/SEEP.1996.534026

Yin, R. K. (2009). *Case study research: Design and methods*. Thousand Oaks, CA, USA: SAGE Publications.

BIOGRAPHIES

Aileen Joan Vicente is an Assistant Professor at the Department of Computer Science, University of the Philippines Cebu. She received her Master's Degree in Information Management from Ateneo de Manila University in 2008. Her research interests include Data Mining, Data Analytics, Natural Language Processing and Computer Science Education. She has been teaching Software Engineering to undergraduate Computer Science students for the past 15 years. She worked as a software developer for a local garments industry before she joined the academe in 2002. Even in the academe, she was assigned administrative tasks involving systems development and management for academe-specific information systems.

Tiffany Adelaine Tan is Assistant Professor (7) and Dean of the School of Management with a specialization in Services Marketing and Management at the University of the Philippines Cebu. Her teaching areas are in general management, organizational behavior, marketing management, branding, brand asset management, advertising, and services marketing. She received her Ph.D. in Business Administration from University of the Philippines Diliman in 2013. Most of her research centers on improving the knowledge, performance, and evaluation of the hospitality industry in the Philippines, i.e., hotels and restaurants. Her primary research interests are organizational commitment behavior, services failure and recovery, and customer satisfaction. Seven of her articles have been accepted for publication in peer-reviewed local and international journals. Before joining the academe, she was in the sales and marketing profession and then became a trainer and consultant.

Alvin Ray Yu is an Instructor of the Department of Computer Science at the University of the Philippines Cebu. He finished his Bachelor of Science in Information Technology degree at the University of San Carlos. He had five years of experience in the IT industry as software engineer, spearheading automation initiatives, developing niche software, and mobile applications before joining the academe. He is currently taking up his Master's Degree in Computer Science in the University of the Philippines Cebu.