Evolving Paper-Based Activities Approach (EPAA) to Promote Interest in Software Engineering Education

SHAHIDA SULAIMAN[1], SARINA SULAIMAN[1], SHARIFAH MASHITA SYED-MOHAMAD[2], WAHIDAH HUSAIN[2]
Faculty of Computing[1]
Universiti Teknologi Malaysia
81310 Skudai, Johor
MALAYSIA
*shahidasulaiman@utm.my, sarina@utm.my
School of Computer Sciences[2]
Universiti Sains Malaysia
11800 USM, Penang
MALAYSIA
mashita@cs.usm.my, wahidah@cs.usm.my

*corresponding author

Abstract: - Software engineering education is vital as an introductory course in computer science or information technology undergraduate programmes. However, it seems to be dull to some educators to teach the concepts as compared to teach courses like programming and database. This phenomenon causes educators to have lack of interest in teaching and in turn affect the interest of learners to grasp the concepts better and relate it with other courses in computer science or information technology. This paper proposes an evolving paper-based activities approach (EPAA) to promote interest in software engineering education among both educators and learners. The approach aims to make software engineering education to be more interesting, engaging and integrated so that learners can appreciate why they learn software engineering course in computer science or information technology programmes. Two groups of students who took the related courses gave the positive feedbacks that the approach increased their interest in learning software engineering mainly in understanding the concept in object-oriented analysis and design using Unified Modeling Language.

Key-Words: - Engineering education; object-oriented analysis and design; Unified Modeling Language

1 Introduction

Software engineering education (SEE) has gained a lot of attention in computer science or information technology programmes. Related schools or faculties in universities normally offer software engineering either as a programme itself or as a specialisation of a computer science or information technology programmes. Over the years, the demand in SEE among undergraduates has increased in tandem with job opportunities for software engineering graduates [1].

However, SEE seems to be theoretical in nature that covers the topics based on stages in software development life cycles (SDLC) mainly planning, analysis, design, implementation, and maintenance. Common courses that universities offer as recommended by IEEE CS and ACM [2] include Project Management (PM), System Analysis and Design (SAD), Software Design and Architecture (SDA) and Software Quality Assurance and Testing (SQAT). The practical part may include the introduction of Computer Aided Software Engineering (CASE) tools in relation to the courses either a complete suite or workbenches (commercial or research prototype tools) for each stage. However, educators can meet learning objectives without even introducing any tools, as the main concerns is to understand concepts or theories. The practical parts mostly cover related courses such as programming and databases. This makes SEE seems to be less attractive among educators and learners.

With the emergence of object-oriented (OO) paradigm and the ubiquitous modelling language that is Unified Modelling Language (UML), SEE mostly adopts OO paradigm in related courses such as SAD and SDA. OO that promotes abstraction requires certain level of thinking among learners to grasp the concepts. The complexity increases with the introduction of many types of views provided by

UML in representing the abstractions using many notations. This aspect causes SEE to be even more difficult in both teaching and learning process.

Therefore, there is a need to promote a better approach in teaching SEE in order to make it becoming more interactive and engaging. There are many works that propose better approach in teaching OO concepts [3][4][5][6][7][8] that mainly focus on OO programming. Some works adopt Problem-Based Learning (PBL) [4][9] besides some works focus on UML [10], requirement engineering [11], software design [12] and at the broader aspect is in software engineering [13]. There is still lack of attention in the issues related to object-oriented analysis and design (OOAD) that are mostly covered in SAD and SDA courses. Hence, this paper proposes an evolving paper-based activities approach (EPAA) that aims to make SEE to be more interesting, engaging, and integrated to both educators and learners.

The organization of this paper is as follows. Section II describes the proposed approach (EPAA), Section III reports the two case studies, and students' feedbacks involving SAD and SDA courses, Section IV explains the related work in SEE and finally Section V concludes the work.

2 Evolving Paper-Based Activities Approach (EPAA)

Evolving Paper-Based Activities Approach (EPAA) has the following main objectives in teaching and learning OOAD related courses using UML notations:

- Increase the interest among educators in teaching such courses that in turn increase the interest among learners.
- Promote the interest among learners in learning such courses that in turn increase the capability of learners in grasping the theoretical concepts.

EPAA requires educators to be creative in using papers of different colors in A4 size during the activities. This means they must plan to spend at least fifty percent of their lectures for the activities and provide a problem that is applicable throughout the semester. The problem must fit to the requirements in the whole activities. In addition, the activities must be relevant from one topic to another using the given color papers that continuously accumulate from one activity to another.

At the end of the course, students will have a big picture of what they have learned in relation to the activities. Fig. 1 shows the overview of EPAA. It suggests that each topic of a course to include a number of related activities depending on the extensiveness of the coverage in each topic.

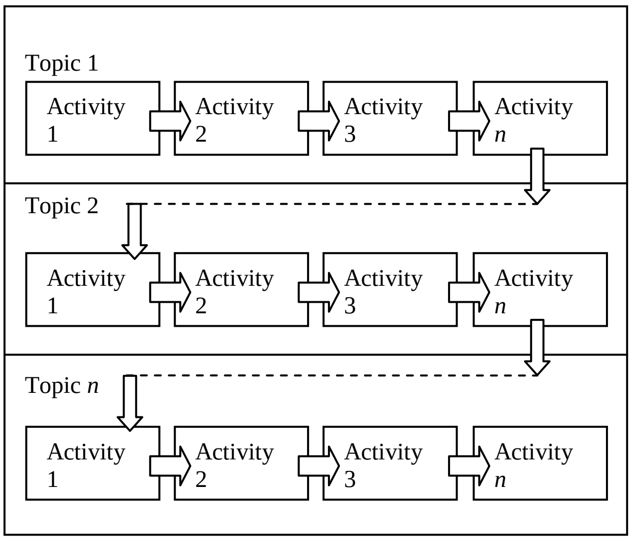

Fig. 1: The overview of EPAA

3 Case Study

This section includes the two case studies that adopted EPAA. It covers two courses: SDA and SAD conducted at the School of Computer Sciences, Universiti Sains Malaysia (USM).

3.1 Case Study 1: Using EPAA in SDA

The first case study involved second year students under Bachelor of Computer Science (Hons.) in Semester II, Session 2009/2010 who took SDA course. Twenty-eight students who must take the 3-unit course under software engineering specialization participated in the study.

Appendix A summarizes the examples of activities conducted in SDA during approximately fifty percent of each lecture hour. Lecturers must conduct all activities as a group work of two to three members. This is ideal when such group is located in a small tutorial room with round tables.

Fig. 2 shows the students working on the color papers given by the lecturer to meet the requirements in each activity. The lecturer took the attendance of the students to ensure they participated in all activities set in the case study. This could eliminate the threat in the analysis as we could ensure all students really participated as a group.

However, it is still possible to adopt EPAA in a bigger group of lectures such as in the case study 2 that is in the next sub-section.

Fig. 2: Students working in groups for each activity during lectures

Each activity relates from one another. After each lecture, students must keep the color papers used in a particular activity to be further refined and referred to in the following activity. As each student must suggest a subsystem, they were responsible to create their own related tasks during each activity. This will promote active participant from each member of a group as they have a subsystem to work on once they move to Topic 2.

This approach inculcates a team building culture among students that are important in software industry. At the same time, they learn from each other and co-operate in completing the tasks. Co-operation is a key mainly in this course. For example in Activity 2.2: Architectural Style students should suggest two possible packages to group classes in each subsystem. Then they have to determine their relationship based on data flow.

This activity requires students to co-operate in order to determine the relationships among different packages of different subsystems. Hence, this also promotes communication skill among students, as they have to discuss and project their ideas among group members. In fact, this approach allows students to simulate a real industry need in communication aspect. This is also one of the ways to encourage students to speak at least in a smaller audience before they present to the whole class.

Thus, the outcome of each activity will be evolving throughout the semester. These activities equipped them with similar structure of requirements in their group and individual assignments. Indirectly, educators can identify whether students manage to grasp the theory taught or not that they should further adopt when doing the given assignments. Fig. 3 is an example of the final outcome from all the activities.

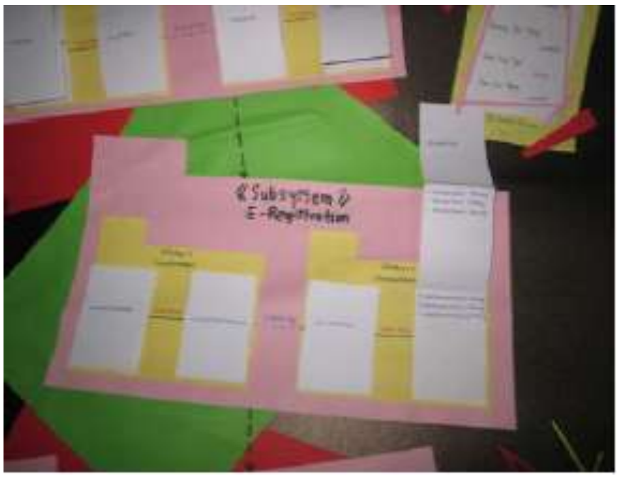

Fig. 3: Students' group work submitted based on series of activities

For the course, there was compulsory lab slot allocated every two weeks to expose students with implementing components using Java. The Integrated Development Environment (IDE) chosen was NetBeans. There were two assignments: (i) Designing and architecting software and (ii) Using Beans in Java Server Page (JSP).

The first assignment required students to choose from the given suggested topic of projects. Based on the chosen project, they indicated the title of the project, its background, objectives, expected benefits, and impact to community. Students must suggest two to three subsystems (each member should work on one subsystem), draw its class diagram and architecture diagram. Then they proposed software structure, its architectural style, and design pattern (one structural and one behavioral) to solve any part of design problem in the project.

In the second assignment, each student must implement individually the proposed subsystem by creating Bean in JSP. They must continue from the same proposed project in the first assignment. Each team member produced a prototype by implementing the selected package in the proposed subsystem under each member's responsibility. Students were able to refine their design to allow they learn to correct their proposed design during the prototyping process. They must develop Bean components and the corresponding JSP files for the chosen package with the minimum two related classes. Then they should explain how to reuse the Bean components in future.

The first case study shows that EPAA managed to increase the interest among educators despite that SDA course is highly theoretical. This is because EPAA suggests fifty percent of the lecture should be activity-based. Thus, lecturers reduce one-way talking, instead they can have more break during lectures by letting the students to have discussion among them during the activity sessions.

This in turn increases the interest of learners in understanding each topic and how they solve the given problem, relate the solution with the activity, and further relate them with future activities in the following topics. The increase of interest among learners creates a more conducive environment for them to sharpen their capability in grasping the theories in such theoretical courses in a more engaging way. Besides, through such activities, educators can detect students' level of understanding in each topic during the activities. Hence, educators can assist them much earlier which in may assist students to complete their assignments as required.

In addition, the group-based activities could increase students' confidence in giving their ideas and solutions to others. Such activities also promote teamwork spirit among students besides learning from each other's feedback and opinions. This is vital in real software industry setting.

3.2 Case Study 2: Using EPAA in SAD

The second case study involved second year students under Bachelor of Computer Science (Hons.) of four-year programme in Semester II of Session 2010/2011. Previously, the programme was in three years as in the first case study. The subject was the first cohort of the four-year programme intake who should take SAD as the common core course.

The case study involved a 2-hour guest lecture as the main researcher or the main author of this paper did not teach the course in the stated semester. Fifty-five students attended the guest lecture for the case study. The students were in the second half of their semester that means the contents discussed during the guest lecture were not new to them.

Thus, the activities planned were quite different from that of the first study with the main aim to pick certain topics that are crucial in SAD and then receive students' feedbacks on the approach at the end of the guest lecture. Table 1 summarizes the activities involved both individually and in-group together with the instructor's note. During the lecture, other support materials related to SAD were available to make students understanding the whole picture of system analysis and design. Hence, the activities would complement what they derive from the materials theoretically and how they could remember them through EPAA.

Table 1: Activities for SAD Course

Activity	Instructor's Note
Activity 1 (5 min. individually): Describe Design using 10 words – Draw a mind map on a blank paper.	Every student has its own interpretation – highlight design is a creative activity. Ask how many has "creative" as one of its descriptions.
Activity 2 (5 min. individually): Flip an A4 color paper into 2 parts vertically, and into 3 parts horizontally. There will be 6 parts. Cut into 6 parts. For each part, write the phases in software development life cycle. Then arrange them in sequence.	Highlight to students the naming could be different in different reference books but the basic must be PADIM (Planning, Analysis, Design, Implementation, Maintenance).
Activity 3 (20 min. in group): Flip an A4 color paper into 2 parts vertically, and into 3 parts horizontally. There will be 6 parts. Cut into 6 parts. Each group member must take 2 to 3 parts depending on the number of members. For each part, flip into 3 parts. Draw a line on each flip mark in order to get 3 divided parts.	Students should know why the 3 divided parts are important to represent different types of view in analysis and design stage as the communications to stakeholders of a system.
Based on the project suggested, each member must think of an object and write the class name in the 1st part, attributes in the 2nd part and methods in the 3rd part. The classes must be different among group members. Then, indicate how each class can relate with other classes. Place all classes on a blank paper. Draw the relationship.	Students should know how to link all classes (objects sending messages to get service).
Now take an A4 color paper of different color than that of classes. Cut the top part to get a folder-like paper horizontally. This folder represents the package.	Students should know how to name the system. Expose students to subsystems and its needs. Highlight the importance of a package diagram.

In the case study, students should sit with their group based on the group assignment given by the course lecturer. The individual activities attempt to study basic understanding of SEE among students before starting with the group work using EPAA. In this study, students used the problem that they had to solve in their project assignment in the course. This could save the time as compared to starting with a new problem. Hence, this study could detect their understanding and misunderstanding in solving the earlier given problem.

In addition, students had to answer two quizzes that could test their knowledge on SAD from the first half of the semester. The quizzes were conducted in between the activities. The first quiz tested on the basic concepts in static model that are class, object, and use cases. The second quiz tested their knowledge in dynamic model focusing on sequence diagrams.

As the group size was quite large and the lecture was in a big lecture hall, there was a limitation in term of space to work on the given papers during the activities. However, having a group of two to three members sitting together during a lecture would enable educators to conduct EPAA even in a large group, in a large venue.

Educators can make a round of check during each activity and interact with students. However, having a larger group reduces the amount of interaction between educators and students. Nevertheless, students still gain the same mood of learning as compared to smaller groups based on the observation made between the first and the second case studies.

The possibility of conducting activities in a large lecture is a good idea among educators mainly in such courses that involve many theories. Lecturers normally read their lecture slides in a huge lecture hall and stuck to the wired microphones. Hence, the fact that group-based activities could enlighten educators' teaching process and students' learning process should motivate more educators to teach theoretical subjects as compared to database and programming courses.

Fig. 4 shows an example of how students could work together during an activity session in a lecture hall. This reflects that group-based activity is still possible in a bigger size of lecture even though it would not be as effective as those in a small group as in the first case study.

Fig. 4: Students sitting in a group of three members in the lecture hall

The feedbacks derived at the end of the lecture attempted to study students' perspectives before and after the lecture in term of the benefits of SAD, appreciation of the role of SAD in SDLC, its relevance with the other courses of the same level involving programming and database courses. Students also gave the feedbacks in term of the benefit in the guest lecture, what do they like most in the lecture, whether or not the activities could help them understand the concepts, and suggestions for improvement the lecture.

Fig. 5 illustrates that there is at least 25% of students could only see the benefits of SAD after this 2-hour lecture while 7% stated "not sure". This probably reflects the fact that students require different approach of teaching in order to show the benefits of mainly in such theoretical courses. This indirectly reflects that educators should open students' mind in order for the students to appreciate why they study certain courses by stressing on the benefits. This will also increase students' interest in learning theoretical courses, which they should link with other practical courses like database and programming.

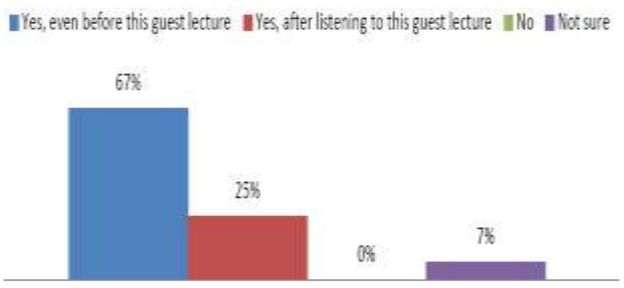

Fig. 5: Knowledge on the benefits of SAD

Fig. 6 shows the balance between those who can appreciate the role of SAD before and after listening to the guest lecture, while 2% still could not appreciate and 7% were not sure. The feedback indirectly deduces that in fact more students did not appreciate the role of SAD only until a different approach of teaching used in making them understanding the concept and its role as the whole in SDLC. Once they can appreciate the role, then they can be more realistic in studying the course that in turn can promote their interest and increase their understanding. More importantly, when educators can increase learners' interest, the educators will be motivated to teach theoretical courses.

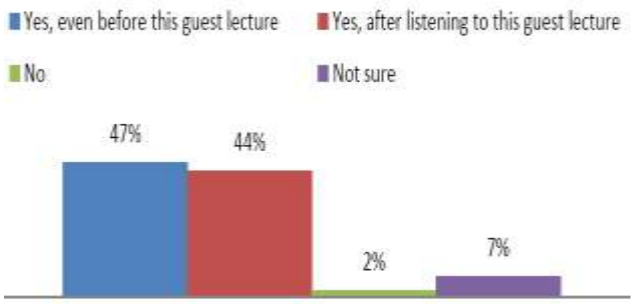

Fig. 6: Appreciation towards the role of SAD in SDLC

Fig. 7 indicates that less than 50% of students could see the relevance of SAD in other related course that is Java development course. In addition, 17% stated "no" while 15% stated "not sure" even after the guest lecture. This reflects the fact that educators need to explain the big picture so that students can relate one course with another under the computer science or information technology programme. This feedback is also expected as the activity did not cover up to transforming the diagram to implementation that could relate with the Java development course.

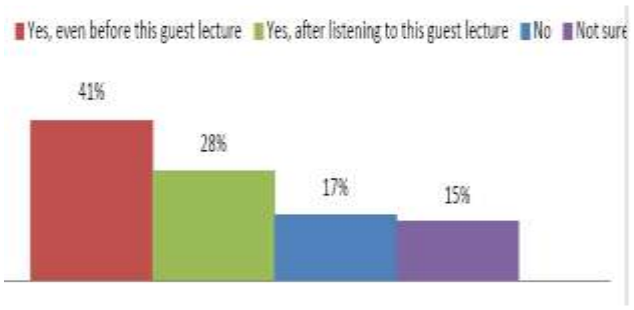

Fig. 7: See the relevance between SAD and Java development course

The next question on the relevance of database course with SAD, Fig. 8 depicts that more than half of the students (67%) could see the relevance even before the guest lecture. On the other hand, 22% stated only after listening to the lecture. Surprisingly, 5% stated "no" while another 5% stated "not sure". Indirectly, the findings show that related activities with appropriate explanation to recall students' experience in other related courses would be helpful to promote better understanding. Hence, students can appreciate more that they should adopt the theories they learn in SAD when doing the practical in the related courses such as the database course.

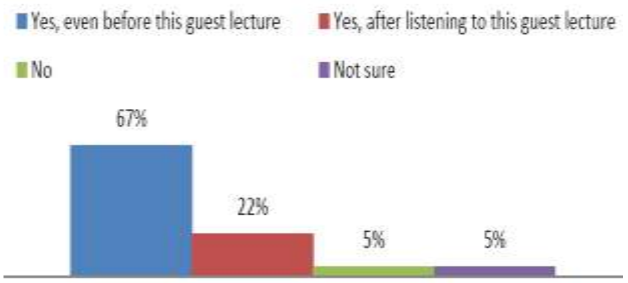

Fig. 8: See the relevance between SAD and database course

From Fig. 9, it shows that the majority or 87% of the students believed that they gain the benefits of the guest lecture. 4% stated "no" while 9% stated "not sure". With more activities in such courses, those under the category of "no" or "not sure" can be eliminated.

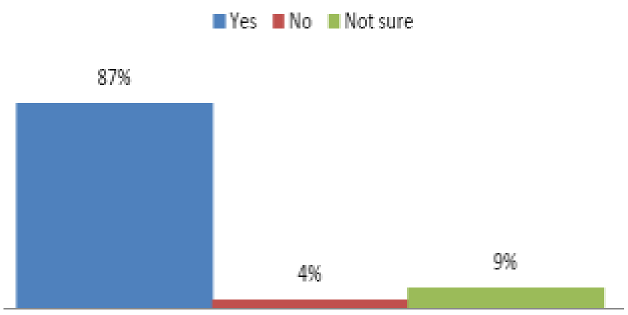

Fig. 9: Benefits of the guest lecture

Table 2 and 3 provide the subjective feedback from 18 students (out of 55) with regard to what the students liked most in the lecture and their feedback on whether the activities could help them in understanding the concepts respectively.

Table 2: Students' feedback on their interest in the lecture

What do they like most in the lecture
Appropriate examples.
I was able to stay awake throughout the lecture.
The activities were fun.
Creative.
Friendly.
The way the lecturer interacted with the students.
The interaction with students.
The interactive way of teaching instead of just reading from the slides.
The differences of teaching method.
Creative lectures.
Quiz 1. Because I know how to do.
Learning in an interactive way.
Interesting, unusual approach, analogy understanding.
A very interesting way to teach SAD.
She provides the creative way to conduct the class.
It's likely with interactive activities.
Yes, I like the creative way in teaching by the lecturer.
About Quiz 2, it makes me clear about the sequence diagram.

Majority of the students liked the interactive activities and the different approach used during the lecture, which they referred as creative, friendly, and interesting. From all the feedback, four students gave the response as "no", "depends", "not so sure", and "not really" when asking whether the activities could help understanding the concepts (see the last four feedbacks in Table 3). The rest including those not listed in the table stated "yes".

Considering the guest lecture was in two hours, the positive feedbacks received from the students reflected that students showed more interest in SAD as the approach used could help them understanding the concepts better. This is of course with the support of other materials such as the big picture of analysis stage and the two quizzes that could help them to make self-assessments upon receiving answers during the lecture itself.

Table 3: Students' feedback on their experience in using EPAA

Whether the activities could help understanding the concepts
Yes, good use of appropriate examples.
No comment.
Yes, understood.
Yes, I can see clearly the concepts from the activities.
Yes, I can apply the activities in the system that I want to do.
Yes, the lecturer summarized the important parts.
Yes, interactive is very impressive.
Yes, I can have a better and clearer picture of the concepts.
Yes, because I implemented it on the spot.
Yes, simplify the concepts to give an overall understanding.
Yes, the activities link the concepts together to show the overall big picture.
Yes, because we learn through "physical activities" and it is quite interesting instead of sitting and listening to lectures.
Yes, because the concept of "do and learn".
Yes, I can pay more attention because of the visual and analogies.
No, the instruction and explanation unclear.
Depends on the activity as some only refresh memories.
Not so sure, cannot get what the lecture try to conceive.
Not really, because I think that giving a detail explanation would be better.

4 Related Work

The study in how to improve teaching in computer science mostly focus on programming classes specifically to learn the concept of object-oriented programming [3][4][5][6][7][8]. However, there is still lack of motivation to improve teaching in theoretical courses such as SAD and SDA.

Many studies also report the use of PBL such as those in teaching object-oriented concept [4] and programming [9]. It still focuses on practical course and does not highlight how to increase interest in theoretical courses in particular. However, PBL is very labour intensive. Thus, it is not suitable for a large group of students.

There are also works that concerns on UML such as that of Nasar [10]. Some works use tools such as Alice [8] that is more suitable in teaching programming but such tools may not be suitable for theoretical based lectures such as SAD and SDA. In addition, Rosca [11] focuses on an active/collaborative approach in teaching requirement engineering that focuses on analysis scope only. While Jia and Tao [12] focus on teaching software design using a case study.

The recent work by Jia [13] reports how to use a case study in teaching the broader scope that is software engineering itself. Our proposed work focuses on OOAD mainly UML diagrams to teach SAD and SDA in either small or large group of students.

Although a numerous work promotes e-learning nowadays [14][15] and the way to evaluate such work [16], paper-based teaching and learning is still relevant especially when we have activities during lectures. This is vital, as theoretical courses mainly comprise mass lectures that do not involve any hands-on which could cause boredom among students. Most importantly, educators should also have the interest in teaching such courses, which actually involve a lot of engagement in real industry environment. For instance, system analysis involves communication with users and other stakeholders while design mainly requires communication among development team members.

Hence, our work aims to highlight the need for such innovation in teaching despite the trend in e-learning to teach engineering-based subjects that require a lot of visualisations and diagrams using computers. This highlights that creativity does not limit to when we have computers and the Internet in front of students. Instead, we could make students becoming more engaging through a simple approach of teaching called EPAA in the selected software engineering related courses that are commonly not favoured among both educators and learners.

5 Conclusion and Future Work

We propose a simple approach using papers that should evolve based on group activities from one topic to another in a semester lecture either in a small or large group of students. The motivation of the study was due to the lack of interest in teaching courses in SEE that are mostly theoretical among educators that in turn affect students' interest and appreciation in learning such courses. The proposed simple approach is called EPAA.

We evaluated the approach in two case studies of a small and large group respectively. Both case studies received positive feedback from the students that we expect could meet our objectives to increase the interest among educators in teaching such courses that in turn increase the interest among learners, and to promote the interest among learners in learning such courses that in turn increase the capability of learners in grasping the theoretical concepts. Hence, the reported case studies could be a motivation among educators to adopt EPAA in their teaching in order to increase their interest in teaching theoretical courses. In fact EPAA is not limited to only software engineering courses. The concept in this approach can also be adopted in any theoretical courses not limited to computer science or software engineering field.

Future work may include conducting an empirical evaluation to see how much EPAA could improve students' understanding in term of speed and correctness in answering related questions as compared to a control group. This may include specific study among selected educators of software engineering related courses.

6 Acknowledgement

We acknowledge the participation and feedback from the students in both case studies.

References:

[1] E. Wulhost, "Survey: Software engineers top list of best jobs", *Money Magazine*, April 13, 2006.

[2] IEEE CS and ACM, Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, Aug. 23, 2004.

[3] D. J. Bagert, B. A. Calloni, "Using an Iconic Design Tool to Teach the Object-Oriented Paradigm", *Proc. 27th Teaching and Learning in an Era of Change, Frontier in Education Conference*, 1997, p. 861.

[4] J. Ryoo, F. Fonseca, D. S., Janzen, "Teaching Object-Oriented Software Engineering through Problem-Based Learning in Context of Game Design", *Proc. IEEE 21st Conference on Software Engineering Education and Training*, 2008, pp. 137-144.

[5] J. Ryoo, F. Fonseca, D. S. Janzen, "Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design", *Proc. IEEE 21st Conference on Software Engineering Education and Training*, 2008, pp. 137-144.

[6] I. Douglas, "Instructional Design Based on Reusable Learning Objects: Applying Lessons of Object-Oriented Software Engineering to Learning Systems Design, *Proc. 31st Annual*

Frontiers in Education Conference, 2001, Vol. 3, pp. F4E-1-F4E-5.

[7] K. Johnsgard and J. McDonald, “Using Alice in Overview Courses to Improve Success Rates in programming I”, *Proc. IEEE 21st Conference on Software Engineering Education and Training*, 2008, pp. 129-136.

[8] R. Klump, “Understanding Object-Oriented Programming Concepts”, *Proc. IEEE Power Engineering Society Summer Meeting*, 2001, Vol. 2, pp. 1070-1074.

[9] E. El-Sheikh, “A Conceptual Problem-Based Learning Environment for Teaching Introductory Programming”, *33rd Annual Frontiers in Education (FIE 2003)*, 2003, Vol. 1, p. T4C-28.

[10] M. Nasar, “VUML: a Viewpoint Oriented UML Extension,” *Proc. 18th IEEE International Conference on Automated Software Engineering*, 2003, pp. 373-376.

[11] D. Rosca, “An Active/Collaborative Approach in Teaching Requirement Engineering”, *Proc. 30th Annual Frontiers in Education Conference*, 2000, Vol. 1, pp. T2C-9-T2C-12.

[12] Y. Jia and Y. Tao, “Teaching Software Design Using a Case Study on Model Transformation”, *Proc. 6th International Conference on Information Technology*, 2009, pp. 702-706.

[13] Y. Jia, “Improving Sofware Engineering Courses with Case Study Approach”, *Proc. 5th International Conference on Computer Science and Education (ICCSE)*, 2010, pp. 1633-1636.

[14] R. Din et al., “Hybrid e-Training Assessment Tool for Higher Education”, *WSEAS Transactions on Advances in Engineering Education*, Issue 2, Vol. 9, April 2012, pp. 52-61.

[15] M. Teichmann and J. Ilvest Jr., “Human Factors Engineering: Digital Teaching Tools and Paper-Free Handouts for Lecture Notes”, *WSEAS Transactions on Advances in Engineering Education*, Issue 2, Vol. 9, April 2012, pp. 31-41.

[16] R. Rahamat et al., “Measuring Learners’ Perceived Satisfaction towards e-Learning Material and Environment”, *WSEAS Transactions on Advances in Engineering Education*, Issue 3, Vol. 9, July 2012, pp. 72-83.

Appendix A: Activities in EPAA for SDA Course

Topic and Activity in Group
Topic 1: Software architecture Activity 1.1: Software Architecture • Ask students to list the facilities of a large web-based application that they have been using. • Identify the architecture and then draw the architecture diagram. Activity 2.1: Architectural View • Represent the architecture of the same web application using Kruchten 4+1 views. • Use UML notation to represent the process view for one of the facilities provided.
Topic 2: Software structure and architecture Activity 2.1: Architectural Structure and Viewpoint • Assume students have to develop a new system to replace the existing Web application. • Indicate 2 to 3 subsystems (each student suggests one subsystem of two classes). • Represent software architecture including the classes. • Explain the software structure in term of platform, programming language, database, COTS. Activity 2.2: Architectural Style • Suggests two possible packages to group classes in each subsystem. • Determine their relationship based on data flow. • Choose the best architectural style and draw the diagram. Justify the choice.
Topic 3: Software design issues Activity 3.1: Identify Design Errors • Identify attributes and operations of each class. • Identify relationships among classes in a subsystem and inter-subsystems. • Indicate the data to be passed among the classes. • Find any flaws or errors in the design then exchange with the other group to check. Activity 3.2: Identify Types of Classes • Observe the proposed classes. • Indicate whether they are concrete nouns or abstract nouns. Activity 3.3: Coupling and Cohesion • Indicate any classes without any relationships with other classes. • Indicate any classes within a package that have any relationship among them. • Indicate any classes in a subsystem that have any relationship with other class in other subsystem. • Identify which relationships are coupling or cohesion. Activity 3.4: Design Principles • Based on each description referring to the Web application, indicate the design principles. Activity 3.5: Control and Handling of Event • Identify a transaction that involves control and handling of events. • Draw the state machine diagram.
Topic 4: Software design quality analysis and evaluation Activity 4.1: • Create a scenario in which an internal user discussed with the developer informally about the Web application beta version. • Discuss lessons learned.
Topic 5: Software design notations Activity 5.1: • Provide attributes and operations of each class. Activity 5.2: • Choose a class and then identify responsibilities and collaboration using CRC card.
Topic 6: Software design strategies and methods • Use stepwise refinement strategy in designing the Web application.