Comparative Critiquing and Example-based Approach for Learning Client-Server Design

Nur Amirah Amjath Jamal and Norhayati Mohd Ali
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
Serdang, 43400 Selangor Darul Ehsan, Malaysia
nur_amirah_1992@yahoo.com, hayati@upm.edu.my

***Abstract*—Software architecture course is one of the courses that are offered in computer science and software engineering programme. This course is related to software architecture design styles and making important design decisions for software design and development process. Several studies reported that students have difficulty in learning and understanding software architecture course. Teaching students about software architecture design and to equip the students with architecture design skills is very challenging. Thus, a comparative critiquing and example-based approach into learning client-server design architecture is proposed in this paper. The main idea is to support students' learning in software architecture design, specifically in the client-server design style. The aim of this paper is to describe the use of comparative critiquing and example-based approach in learning client-server design style. We believe that this approach would enhance the students' learning and understanding in client-server design. Furthermore, it would blend the learning performed in the traditional classroom setting.**

Keywords—client-server design, comparative critiquing, example-based, design critiquing tool, software architecture design

I. INTRODUCTION

Garland and Shaw [1] claimed that an effective software engineering requires facility in architectural software design. Numerous studies explain the importance of software architecture in the design and development of software [1-4] Some of the importance are making the right and significant design decisions, recognizing common paradigms so that high-level relationships among systems can be understood and getting the right architecture that could lead to the success of a software system design [1]. The design decision on the software architecture is crucial because of its effect on the development of the system.

Realizing the importance of software architecture design, most universities that provide Computer Science and Software Engineering programme will offer the Software Architecture course to their undergraduate and graduate students. The offering of this course is to equip the students with knowledge and skills in software architecture design and they can apply it in software industry. However, several studies report the challenges of teaching software architecture design course [2-6]. Teaching and learning of software architecture design course at universities requires an appropriate guidance from the educators (e.g., instructors, lecturers, teachers, mentors and others). There are many approaches that can be used in improving the teaching and learning process. These include a computer-supported learning tool using a critic-based approach and example-based approach, which have been recognized, in the education area.

The aim of this paper is to describe the development of a client-server design critic tool (CSDCT) to assist the students in the learning of software architecture design course, specifically the client-server design topic. The focus of client-server design critic tool is to provide feedback to students regarding the architecture designs they created. We first introduce the background and motivation for this research. We then explain our comparative critiquing and example-based approach. The implementation and evaluation of CSDCT are then described. Finally, we conclude with future work.

II. BACKGROUND AND MOTIVATION

A. Educational Critiquing Tool

Design critic concept is not new and work on critiquing tools/systems has been published by several researchers. Exemplars of educational critiquing tools can be obtained from previous works [7-10]. The critic-based approach concept is mainly to identify potential problems; provide suggestion and possibly offer automated or semi-automated artifacts improvements to users [11]. Computer-supported learning tools using a critic-based approach have been recognized as an essential supporting tool in an education environment.

For instance, the UMLGrader [7] tool, an automated class diagram grader developed to facilitate students in learning to draw UML class diagrams. The tool provides instant and automated feedback to students by comparing the student's diagram against the specified solution [7]. Similarly, the Essay Critiquing System (ECS) [8] that provides immediate feedback on essay content and organization for students' revision [8]. Students use the ECS to submit their essay and get feedback regarding the essay content and organization. The ECS system helps the students on essay writing by comparing the students' essay and model essays [8]. Another

educational critiquing tool in the domain of architectural design studio is the Furniture Design Critic (FDC) developed by [9-10]. The FDC [9-10] tool assists the architecture design students in drawing and modeling a design problem. The FDC has a pattern matcher component used to compare the student's design against the stored design constraints to detect any critic violations. The FDC applies a constraint-based design critic program that offers students with five delivery types and three communication modes. The students can obtain feedback from FDC in a form of: interpretation, introduction, example, demonstration and evaluation. The three communication modes are written comments, graphical annotations and images. Details of this work can be referred to [9-10].

The three exemplars above employed a comparative critiquing approach in comparing students' solutions with the predefined/stored solutions in the system. In a comparative critiquing approach [12], complete and extensive domain knowledge is important to generate good solutions. Robbins [12] suggested that a comparative critiquing could direct the users to make their work like the one that the system proposed. Thus, this approach guides the user to a known solution [12].

B. Example-Based Learning

One way to improve teaching and learning process is by providing examples to students, as these examples can demonstrate the difficult concepts that are hard to understand by the students. In addition, the students could use the examples for inspiration and motivation in understanding the learning subjects/courses. A considerable amount of literature has been published on using examples to enhance the teaching and learning process.

For instance, a research by [13], reported that examples can guide students in producing better software designs. Karanesh et.al [13] performed a controlled experiment to evaluate the use of a collection of software design examples. The results showed that the examples support the students in creating and improving their software designs and the students are more confident in producing their designs. Likewise, a study by Van Gog et. al [14] claimed that the use of examples able to reduce the students' cognitive load during the learning process. Van Gog et.al [14] research was to compare the effects of example study, example-problem pairs, problem-example pairs, and problem solving on students' cognitive load and learning. Although the learning domain is about electrical circuits troubleshooting, we are interested in their work because of the use of examples in the learning process. Details of this work can be referred to [14]. An article written by Gulwani [15], stated that human learning and communication normally structured around examples. Understanding certain concepts by using examples would be much easier for students [15]. The use of examples in educational technologies has many purposes. Gulwani [15] reported there are three ways examples are used in computer-aided educational technologies: i) as *input* to express intent, ii) as *output* to generate the intended artifact, and iii) *inside* the underlying algorithms for inductive reasoning. These are explained and discussed in an article contributed by Gulwani [15].

C. Motivation

While educational critiquing systems and example-based learning have been demonstrated effective in providing feedback and improving students' learning [7-10, 13-15], the combination of the two approaches (i.e. comparative critiquing and examples) have not been applied for software architecture design domain in a teaching and learning process.

Some of the challenges related to teaching software architecture design as reported by [2], include students have limited experience and exposure to difficult high-level design tasks, design problems normally starts from scratch due to limited capacity and time, students expect well-formed problems that lead to a good solution, and fuzzy problems are difficult to address in university courses. Other challenges discussed by Mannisto et al. can be referred to [2]. Rupakheti and Chenowet [4] also report similar challenges in teaching software architecture. Rupakheti and Chenowet [4] point out in their experience report, that teaching Software Architecture at any level is not easy. Two main challenges in teaching software architecture course as reported by [4] are: 1) teaching the undergraduate students who have limited experience in the software industry, and 2) educators' background and knowledge that must include various real-world experience in software architecture. However, the teaching would not be difficult for the educators (lecturers or instructors) and the audiences that are involve and have experience in software design tasks [4].

A very limited number of educational tools are available for software architecture design. Most educational tools in software design domain emphasis only on a specific area such as software modelling or software programming. There are several commercial tools that support software design domain. However, these tools usually offer a large number of functionalities and focused to professionals and not to students [16]. Due to the challenges stated above, have motivated us to propose a computer-based supporting tool for learning software architecture design using a comparative-critiquing and example-based approach. However, our research is only focus on the learning of software architecture design specifically on Client-Server design style. The following section explains our project approach.

III. Comparative Critiquing and Example-based Approach

The overview of our Client-Server Design Critic Tool (**CSDCT**) development is shown in Fig.1. The user for CSDCT is students who would like to improve their understanding and knowledge in client-server design architecture. Initially, the students can select the client-server design question and propose their design's answer/solution. Next, a matching and comparison process of the proposed answer/solution by the student with the stored design answer/solution is performed. If the proposed answer is not matched to the stored design solution, a critique will be

The work was supported by UPM via Fundamental Research Grant Scheme, a grant from the Ministry of Higher Education, Malaysia.

displayed. A critique and suggestion is provided by the CSDCT to help the students to do corrections in their design. Based on the feedback, the students can improve their design solutions by submitting another answer and checked again by the CSDCT. With CSDCT, the students can learn what kind of component involves in client-server design styles. These functionalities are illustrated in a use case diagram as shown in Fig.2.

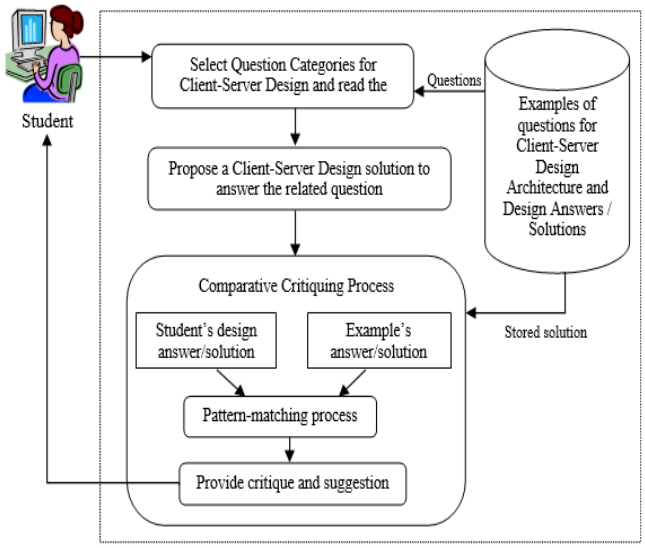

Fig. 1. Overview of CSDCT Development

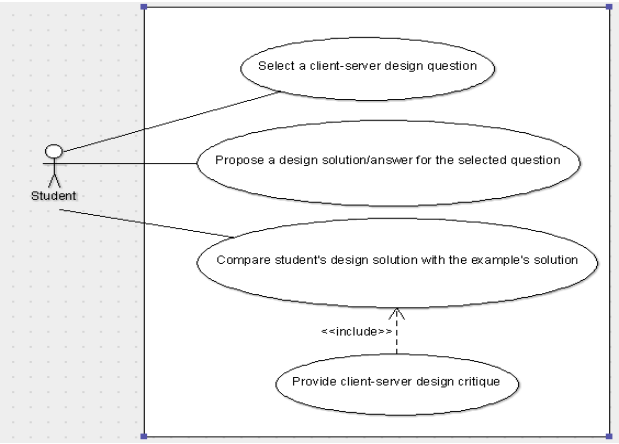

Figure 2: Use case diagram for CSDCT

The development of CSDCT is guided by critic taxonomy [11] that consists of eight groups with several elements. Based on these eight groups, the appropriate requirements for CSDCT are identified. Brief explanation on these requirements is defined in Table 1. The categories of questions for CSDCT are classified into three parts-*Easy*, *Moderate* and *Difficult*. We collected examples of questions and answers for client-server design architectures, and stored in a question and answer repository. If students manage to answer all questions in Easy level, then they can move to the Moderate level and finally the Difficult level. The types of client-server design architecture for CSDCT are shown in Table 2. The following section will explain the implementation and evaluation of our CSDCT.

TABLE 1: REQUIREMENTS FOR CSDCT DEVELOPMENT

Group	Elements	Description
Critic domain	Domain knowledge	Software engineering education (client-server design architecture)
Critiquing Approach	Comparative critiquing	Compare the student's design solution with the example's design solution
Mode of Critic Feedback	Textual	Critique is displayed textually
Critic Realization Approach	Rule-based	Critics are realized using rule-based, If-then-else approach
	Pattern-matching	The properties of the proposed design solution/answer are matched against the properties of example's solution
	Programming code	Critic rules are implemented in the coding
Critic Dimension	Passive critic	Critique will be displayed only after the student submit the proposed design solution/answer
Type of Critic Feedback	Explanation	Explanation is provided as to explain the rationale why the proposed design is incorrect
	Suggestions	Suggestions is provided as to guide user towards the correct/valid design
	Constructive	Feedback in a constructive style is given when user manage to propose correct answers to all design questions.
Type of critic	Completeness	Used in finding whether the proposed design having all the required component and connection
	Structure	Used in checking whether the right component is used and its connection with other component
	Metric	Used in checking the correct number of component and connection is used in a design.

TABLE 2: QUESTION CATEGORY AND TYPES OF CLIENT-SERVER DESIGN ARCHITECTURE

Question category	Types of client- server design
Easy	Normal architecture - distributed database
	Normal architecture - distributed processing
	1-tier architecture
	2-tier architecture
	3-tier architecture
Moderate	Normal architecture
	Architecture include thin client
	Architecture include fat client
	Architecture include thin and fat client
	3-tier architecture
Difficult	Normal architecture with more server component
	Multi-tier architecture or n-tier architecture
	Multi-tier architecture or n-tier architecture
	Typical e-commerce architecture
	Combination of normal architecture, 2-tier architecture and 3-tier architecture

IV. Implementation and Evaluation of CSDCT

Client-Server Design Critic Tool (CSDCT) is an educational critiquing tool for drawing client-server design architecture. It is developed to facilitate students in designing client-server design architecture styles. The intention of this CSDCT development is to improve students' understanding and learning about client-server design architecture. The CSDCT provide automated critiquing by comparing the student's proposed design answer with the standard/stored example's solution. The guidelines or rules of client-server design architecture are used in the automated critiquing to detect design errors made by the students. The CSDCT also provide explanation and suggestions to improve the design errors.

The interface for CSDCT includes CSDCT **main interface** and its **palette**. The main interface design as shown in Fig.3 includes the Question panel, Preview, Diagram space, Select Question panel, Guideline, and Submit Design panel. The description each element is as follows:

- •The question panel: display the question.
- •Preview: enable the user to see the overview of the design drawn in diagram space
- •Diagram space: enable the user to answer the question by drag and drop the components that could be found in the palette (shown in next section)
- •Select question: enable user to select which question they want to answer; each question level namely "Easy", "Moderate", and "Difficult" contains five questions each
- •Guideline: guide user on how to start designing or how to use the prototype
- •Submit Design button: enable the user to check their proposed design for the selected question

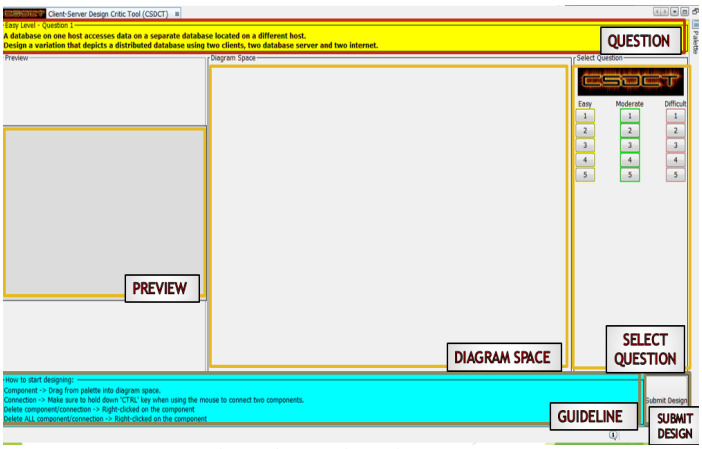

Fig.3. The Interface of CSDCT

The palette area of CSDCT as shown in Fig.4 consists of the component category (client, server, and internet/intranet) of client-server design. When a student attempts to answer a client-server design question, she/he has to use this pallete area to draw a possible design solution. The student will choose the appropriate component listed in the pallete. The component categories are classified as follows:

- Client: client, thin client, fat client, and web browser client
- Server: server, application server, business object server, catalog server, database server, file server, legacy database server, picture server, relational database server, video server, web application server, and web server
- Internet/intranet: internet and intranet

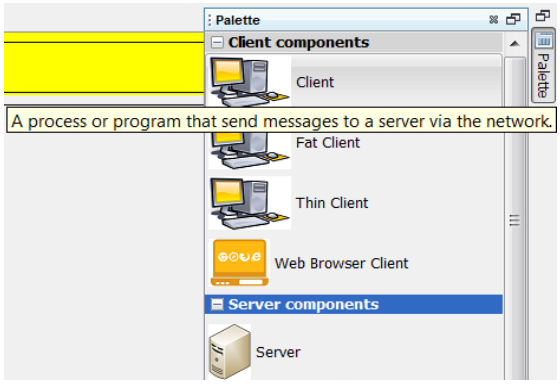

Fig.4. CSDCT Pallete

When a student submits the proposed design answer/solution for the client-server design question, a comparative critiquing will be performed. If the proposed design answer does not matched with the standard/stored design solution, then a textual critique will be displayed to prompt user about the design mistakes in the client-server design. A suggestion as to fix the mistakes is provided to the student and the student can submit another answer. Fig.5 and Fig.6 show the examples of a critique displayed after a student select the 'Submit Design' function. The types of critic for the client-server design architecture are: completeness, structure and metric critics. These critics are related to the connection and components of client-server design. A connection critique will be displayed when the connection between the components is wrong. A component critique will be displayed when the wrong component is selected. These are checked and compared based on the example's solution defined in the system's repository.

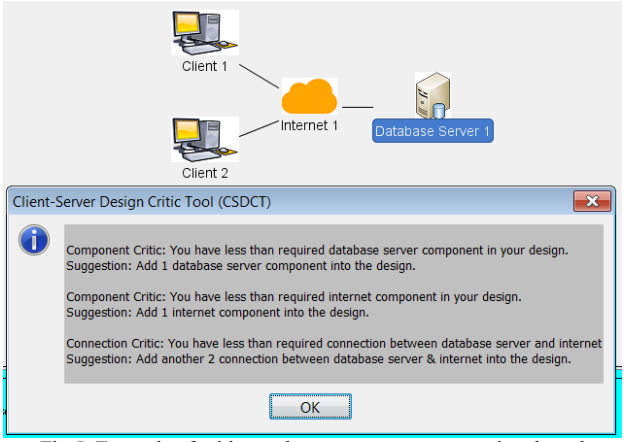

Fig.5. Example of critique when a wrong component is selected

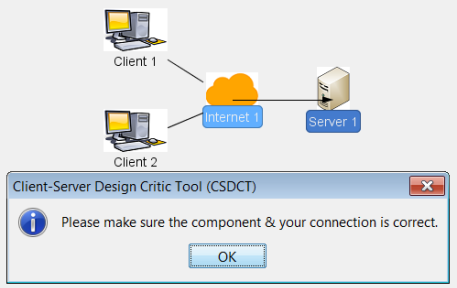

Fig.6. Example of a critique when an invalid connection is created

The CSDCT provides an automated critiquing via comparative critiquing and example-based approach. It provides the students with an easy-to-use tool and facilitates the self-learning process. We have conducted a user evaluation with ten (10) software engineering students who had basic knowledge in software architecture course to assess the CSDCT. Although the respondent size is small, we complied with the general rule suggested by Hwang and Salvendy [17] for usability evaluation: the *10±2 Rule*. Furthermore, Alroobaea and Mayhew [18] and Nielsen [19] claimed that by increasing the number of participants will not affect the usability score since the error encounter within the small number of participants will just be repeated and other participants might encounter the same error. The aims of the evaluation are to assess the effectiveness and usability of the CSDCT. The evaluation is performed via experiment-based approach. The experiment is structured into three parts. Part one and two, provides a task list that need to be performed the participants. Both parts, includes the need to use the CSDCT in drawing a client-server design. Part Three provides a questionnaire that must be answered by the participants once they completed the task list.

Table 3 shows the record of the time taken to complete part one task (i.e. answer Question 1) and part two task (i.e. answer Question 2) by ten participants.

TABLE 3:TIME TAKEN TO COMPLETE QUESTION 1 AND 2 OF MODERATE LEVEL FOR CLIENT-SERVER DESIGN QUESTIONS

Participant	Time taken to complete a question (minutes)	
	Question 1 (Paper-based + use CSDCT)	Question 2 (use CSDCT only)
1	14.47	3.55
2	4.33	1.25
3	7.03	3.1
4	5.54	1.39
5	4.26	2.58
6	5.28	2.38
7	4.48	2.24
8	2.47	2.3
9	3.4	1.56
10	5.27	3.43

The question 1 of client-server design requires the participants to answer the design question using a paper. Then transfer the answer to CSDCT for checking the student's answer with the predefined example solution for the question 1. As for the question 2, it requires the participants to answer the question by using the CSDCT tool only. The participants read the question 2 and directly proposed their answer in CSDCT. The comparative critiquing against student's answer and example's solution is performed. The time taken to perform these tasks are recorded and shown in Table 3. The effectiveness evaluation for CSDCT is assessed by comparing the time taken to complete question 1 and question 2. The questions are from the 'Moderate' level questions. From the Table 3, it shows that the time taken using the CSDCT to answer the client-server design question is less compared using a paper and then checked with CSDCT. The reason for this is because students directly can draw/answer their proposed solution using the listed components and connection provided in the CSDCT pallete. The feedback from the CSDCT is also immediate once the students submit their design answer for checking the correct answer. Thus, we can state that the CSDCT is effective and useful for students to improve their understanding and learning in client-server design architecture.

Table 4 shows the System Usability Score (SUS) used for the CSDCT evaluation based on the responds from ten participants. The System Usability Scale (SUS) is proposed by Brooke [20] and it provides a standardized, simple, ten-item scale which gives a global view of subjective assessments of usability.

TABLE 4. SYSTEM USABILITY SCORE (SUS) FOR CSDCT

Participant	**SUS Score**
1	85.0
2	92.5
3	100.0
4	77.5
5	87.5
6	62.5
7	87.5
8	72.5
9	85.0
10	87.5
Average	**83.8**

From Table 4, it shows that the highest SUS score for CSDCT is 100 while the lowest SUS score is 62.5. The average score of SUS for CSDCT is **83.8**. According to [21] the System Usability Scale (SUS) average score is 68. [21,22]. If the score obtained is above 68, it is said that the system usability is above average; and if the score obtained is below 68, it is said that the system usability is below average. Figure 7 shows the suggested ratings, scores and scales by [21] that could provide clear information based on the obtained average SUS score.

The average SUS score of CSDCT prototype is **83.8**, which indicates *acceptable* in the acceptability range and CSDCT achieved B in the grading scale indicating *Excellent* rating. Hence, we can conclude that the usability of CSDCT is at a satisfactory level.

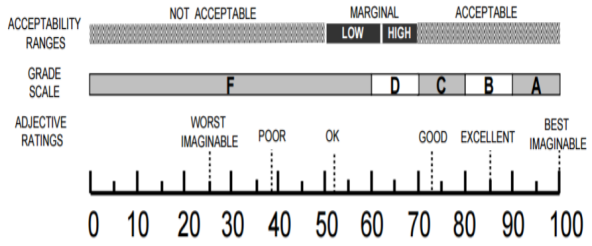

Fig.7: Comparison of adjective ratings, acceptability score and grade scale in relation to the average of SUS score [21]

V. Conclusion And Future Work

We have described our comparative critiquing and example-based approach for learning client-server design architecture. This approach is hoped to support the software engineering/computer science students in learning software architecture design course, specifically the client-server design style. We have briefly presented several educational critiquing tools using comparative critiquing approach and a few of example-based learning in various domains for providing feedback to students. We also explained our project methodology and overview of CSDCT development.

This project work is very much proof-of-concept that a comparative critiquing approach and example-based learning could support the software engineering/computer science students to learn the software architecture design course in a simple way. In addition, a well-designed of educational supporting tool could facilitates students to improve students' learning on their design problems specifically client-server design style. Finally, we expect that utilization of comparative critiquing approach and example-based learning should enhance the self-learning process in understanding the client-server design concepts.

Our plans for future work on this research include the construction of examples of other types of design architecture style such as pipe-filter design, component-based design, cloud-computing design and others. More architecture design critics that reflect the styles of software architecture design. Generally, these architecture design examples and comparative critics can be added incrementally in the repository.

Acknowledgment

The authors would like to thank the software engineering postgraduates' students who have participated voluntarily in the user evaluation survey.

References

[1] D. Garlan and M. Shaw, "An introduction to software architecture.", Advances in software engineering and knowledge engineering, 1(3.4), 1993.

[2] T. Mannisto, J. Savalainen and V. Myllarniemi, "Teaching software architecture design", 7th Working IEEE/IFIP Conference on Software Architecture, 18-21 Feb 2008, Vancouver, Canada, p.117, 2008.

[3] A.I. Wang and B. Wu, "Using Game Development to Teach Software Architecture," International Journal of Computer Games Technology, vol. 2011, Article ID 920873, 12 pages, 2011. doi:10.1155/2011/920873

[4] C.R. Rupakheti, and S. Chenoweth, "Teaching software architecture to undergraduate students: an experience report.", In Proceedings of the 37th International Conference on Software Engineering-Volume *2*, IEEE Press, Florence, Italy, May 16-24 2015, pp.445-454, 2015.

[5] M. Galster, and S. Angelov, "What makes teaching software architecture difficult?", In Proceedings of the 38th International Conference on Software Engineering Companion, ACM, pp. 356-359, 2016.

[6] S. Jarzabek and P.K. Eng, "Teaching an advanced design, team-oriented software project course.", In *18th Conference on Software Engineering Education & Training (CSEET'05), IEEE,* pp. 223-230, 2005.

[7] R. W. Hasker, "UMLGrader: an automated class diagram grader", Journal of Computing Sciences in Colleges, 27, pp. 47–54, 2011.

[8] C. Lee, K.C. Wong, W.K. Cheung, and F.S. Lee, "Web-based essay critiquing system and EFL students' writing: a quantitative and qualitative investigation", Computer Assisted Language Learning, vol. 22, No. 1, pp. 57-72, 2009.

[9] C.S. Kong, H. Ogata, H.C. Arnseth, C.K.K. Chan, T. Hirashima, F. Klett, J.H.M. Lee, C.C. Liu, C.K. Looi, M. Milrad, and A. Mitrovic, "Constraint-based design critic for flat-pack furniture design", in Proc. of the 17th International Conference on Computers in Education., pp. 19-26, 2009.

[10] Y. Oh, and Y. K. Oh, "A computational model of design critiquing", Artificial Intelligence Review, pp. 1-27, 2016.

[11] N. M. Ali, J. Hosking, and J. Grundy, "A Taxonomy and Mapping of Computer-Based Critiquing Tools", *IEEE Transactions on Software Engineering*, vol. 39, no. , pp. 1494-1520, Nov. 2013.

[12] J. E. Robbins, Design critiquing systems, Technical Report UCI-98-41,1998, http://www.ics.uci.edu/~jrobbins/papers/CritiquingSurvey.pdf.

[13] B. Karasneh, R. Jolak, and M.R.V. Chaudran, "Using examples for teaching software design", 2015 Asia-Pasific Software Engineering Conference, pp.261-268, 2015.

[14] T. Van Gog, L. Kester, and F. Paas, "Effects of worked examples, example-problem, and problem-example pairs on novices' learning", Contemporary Educational Psychology, vol.36, pp.212-218, 2011.

[15] S. Gulwani, "Example-based learning in computer-aided STEM education", Communications of the ACM, vol.57, no.8, pp.70-80, August 2014.

[16] E. Ramollari, and D. Dranidis, "StudentUML: An educational tool supporting object-oriented analysis and design." in Proc. of the 11th Panhellenic Conference on Informatics (PCI 2007), pp.363-373, May 2007.

[17] W. Hwang and G. Salvendy, "Number of people required for usability evalaution: the 10±2 rule", Communications of the ACM, vol 53. Issue 5, pp.130-133, May 2010.

[18] R. Alroobaea, and P. J. Mayhew, "How many participants are really enough for usability studies?", *Science and Information Conference (SAI), 2014*. IEEE, 2014.

[19] J. Nielsen, "How many test users in a usability study?" Retrieved from http://www.nngroup.com, 2012.

[20] J. Brooke, "SUS-A quick and dirty usability scale." *Usability evaluation in industry* 189.194, pp.4-7, 1996.

[21] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale." *Journal of usability studies* 4.3, pp.114-123, 2009.

[22] J. Sauro, "A practical guide to the System Usability Scale: Background, benchmarks, & best practices", Denver, CO: Measuring Usability LLC, 2011.