Applying Case-Based Learning for a Postgraduate Software Architecture Course

Eng Lieh Ouh
School of Information Systems
Singapore Management University
elouh@smu.edu.sg

Yunghans Irawan
Institute of Systems Science
National University of Singapore
yirawan@nus.edu.sg

ABSTRACT

Software architecture remains a difficult subject for learners to grasp and for educators to teach given its level of abstraction. On the other hand, case-based learning (CBL) is a popular teaching approach used across disciplines especially in business, medicine and law where students work in groups apply their knowledge to solve real-world case studies, or scenarios using their reasoning skills and existing theoretical knowledge. In this paper, we provide how we apply case-based learning to address the challenge in teaching a postgraduate software architecture course. Our learners are postgraduate students taking a master's program in software engineering. We first describe our design of case-based learning for our software architecture course. We then analyse the survey ratings and learners' profile to evaluate the effectiveness of the proposed case-based design. These data are gathered from 9 class runs over a period of 8 years. Our analysis results show the effectiveness of our case-based design and significant relationships between this effectiveness to the learners' years of working experiences and the number of learners. Key contributions in this paper are our proposed case-based design for software architecture and the analysis findings.

KEYWORDS

Software architecture; curriculum design; case-based learning; pedagogical approach

ACM Reference format:
E. L. Ouh and Y. Irawan. 2019. Applying Case-Based Learning for a Postgraduate Software Architecture Course. In *Proceedings of ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'19), July 15-17, 2019, Aberdeen, Scotland, UK.* ACM, NY, NY, USA. 6 pages. https://doi.org/10.1145/3304221.3319737

1 INTRODUCTION

In a software system, creating an architecture of a system comprises of designing the system's high-level building blocks within the user's environment that can exhibit the required non-functional qualities. These qualities of the system are determined by the design decisions made of an architect. The abstractness of architecture concepts poses a key challenge for learners to grasp and educators to teach.

Galster and Angelov noted in their work [1] that learners often find it challenging for them to appreciate the fuzziness of software architecture concepts and unable to switch their mindset from the concrete aspect of software programming and low-level software design to abstract architectural thinking. In addition to the vagueness of the concept of software architecture itself, Boer, Farenhorst and Vliet also commented in their work [2] that architecture problems are usually "wicked" and requires educational methodologies that deviate from the traditional active lecturer / passive student relation. For one who is used to writing code and compiles to get a deterministic result, the mindset of structuring components together in a diagram, justifying their key decisions with potential trade-offs without a concrete output and there is no one single perfect solution is a frustrating divergence from what they are doing. -

The skill set for one be a competent software architect is also multi-faceted which increase the level of difficulty of teaching it in a classroom environment. The role of a software architects entails one to have the following desired skills:

S1. *Technical skills to design and structure the software components from conceptual theoretical thinking to a practical digital solution.* Software architecture is the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [1].

S2. *Analytical skills to evaluate the problems quickly, analyse the possible root causes and make significant design decisions for the project.* An architect who is unable to make significant design decisions (principles) on the components and their relationships in an environment where much is unknown is unlikely to succeed. "The life of a software architect is a long and rapid succession of suboptimal design decisions taken partly in the dark." [2].

S3. *Learning skills and motivation to keep up to date with the quickly evolving tools and technologies.* Though software architects do not need to be technology experts in all areas, it is essential that an architect is motivated to keeps abreast of the frequently changing technology trends and practices.

S4. *Effective communication skills to understand and negotiate project requirements with relevant stakeholders.* Specifically, an architect should have effective language skills, including

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland UK

ACM ISBN 978-1-4503-6301-3/19/07…$15.00 https://doi.org/10.1145/3304221.3319737

speaking, writing, and presentation abilities to address complex problems with a seemingly simple design that are easy to grasp.

The design of a software architecture course needs to incorporate elements that can effectively impart the above skills. Case-based learning's main traits are that a case is used to stimulate and underpin the acquisition of knowledge, skills, and attitudes. These cases present a disciplinary problem or problems for which students devise solutions under the guidance of the instructor. The following bullet points discuss the benefits of applying case-based learning (CBL) with respect to the desired skills of a software architect.

- (With respect to S1) CBL provides students with an opportunity to see theory in practice. This integration of knowledge and practice in real-world or authentic contexts expose students to viewpoints from multiple sources. Students can also see how a decision impact different participants, both positively and negatively.
- (With respect to S2) During the case discussion, students are required to analyse data and background information in order to reach a conclusion. Since many assignments are open-ended, students practice choosing appropriate analytic techniques as well.
- (With respect to S3) In the midst of searching answers to the case problems, students learn the importance of continuous learning to keep up to date with the solutions available. They will be motivated to search for new architectural styles and technologies for a better solution to the case problems.
- (With respect to S4) In CBL, students can develop communicative and collaborative skills along with content knowledge. In their effort to find solutions and reach decisions through discussions, students sort out factual data, articulate issues, reflect on their relevant experiences and draw conclusions they can relate to new situations. In the process, they acquire substantive knowledge and develop collaborative, and communication skills.

In this paper, we seek to investigate and validate the following two high-level research questions.

RQ1 How do we design case-based learning in software architecture for postgraduate students?

RQ2 How is the effectiveness of our case-based design for our group of learners to learn software architecture?

The rest of the paper is organised as follows: We first describe the related work in section 2. We seek to address RQ1 with our proposed case-based design in section 3 and our implementation of the case-based design in section 4. In section 5, we seek to address RQ2 by explaining our evaluation design followed by our evaluation analysis in section 6. Lastly, we explain the threats to validity in section 7 and conclude this paper in section 8.

2 RELATED WORK

Comparing Problem-Based Learning with Case-Based Learning: Effects of a Major Curricular Shift at Two Institutions: Srinivasan, Wilkes, Stevenson, Nguyen and Slavin in [3] describe their experiences in applying and comparing problem-based learning (PBL) and case-based learning (CBL) methods to the medical curriculum for students studying in two major academic medical centres. The results of their study show that learners and faculty overwhelmingly preferred guided inquiry-based of CBL over open inquiry-based of PBL for their medical curriculum. In our study, we seek to validate the effectiveness of applying CBL and the design elements of a case for a software architecture course.

Teaching adult learners on software architectural thinking skills: Ouh and Irawan [4] present their experiences to teach software architecture course to adult learners and the insights to applying both the case-based (CBL) and problem-based (PBL) methods. Their research shows that learners better appreciate the open inquiry aspect of PBL where they can learn from other learners' experiences and thinking within a short time span. PBL based approach, which does not have any pre-set goals and expected outcomes like CBL, tends to in a way works well with adult learners. On the other hand, there are also learners who appreciate the guided and structured aspects of the workshop based on the CBL method. They suspect that this difference may be due to the level of experiences of their adult learners. The learners' level of experiences of learners in their study on average is slightly higher as compared to the learners' level of experiences in our study. Besides the level of experiences, their study also compares and analyse the effectiveness of case-based learning against the other possible factors of the learner's profile.

Teaching Software Architecture to Undergraduate Students: Rupakheti and Chenoweth in [5] describe their experiences and learnings in teaching software architecture course to undergraduates. Their course design involves daily quizzes, homework, paper reviews and project work. They strongly favour a project-based version of teaching architecture, even at the undergraduate level. Ouh and Irawan [6] adopt an experiential risk learning model to design their software architecture course for undergraduates to address the challenges of teaching abstract software architecture concepts to undergraduates. The model comprises of activities to simulate risks that can happen in practical scenarios, and their role is to be able to recognise these risks, reflect on the causes and mitigate these risks.

3 PROPOSED CASE-BASED DESIGN FOR SOFTWARE ARCHITECTURE

One key artifact in case-based learning is the case, and an effective case design is essential to achieve the benefits of CBL. We review existing work in CBL and samples of software engineering case studies available in [7]. To the best of our knowledge, there is no existing work to discuss the design of a case for software architecture. In this section, we propose our case-based design for

software architecture to address our first research question RQ1. We break down RQ1 into 2 specific sub-questions.

RQ1.1 What is the proposed case-based design for software architecture?

RQ1.2 How to conduct the case-based design for software architecture?

We will address RQ1.1 in this section in terms of case format, case conduct and required elements of a case. RQ1.2 will be discussed in the next section on the implementation of our case-based design.

The format of a case influences how to use it with the students and can be in the form of *bullet-cases, mini-cases and descriptive cases*. *Bullet-Cases* comprises of two or three sentences with a single teaching point, and students discuss them in small groups or individually. This format is used during the lectures and generally conducted in a short duration of 5-15 minutes. These cases enable the students to quickly review key concepts taught, analyse and discuss the potential problems. *Mini-Cases* are designed within a single class meeting and usually tightly focused, useful for helping students apply concept within a practical scenario. We use this format in workshops and usually incur a longer duration from 60 minutes to 2 hours. *Descriptive cases* are written up to 5 pages (1-2 paragraphs per page), and each page is disclosed to the student with discussion and development of learning goals and study questions for each part of the case. This type of cases is typically used for two or more class meetings. We use the descriptive cases for assessments where the students can review the case on a longer duration of days before giving their analysis and recommendations. The students can gain technical and analytical skills when they discuss the cases based on these formats. The descriptive cases also allow the students to learn from other sources due to the longer duration and less focused on a single concept.

The conduct of a case teaching can be in the form of *discussion, debate or public hearing*. *Discussion format* refers to the typical format when the instructor asks probing questions, and the students analyse the problem depicted in the story with clarity and brilliance. *Debate format* is well suited for cases where two diametrically opposed views are evident. *Public hearing* format is an ideal format to allow a variety of people to speak and different views to be expressed. The students can practice their communication and collaborative skills during the case conduct. We use discussion format for bullet-cases during lectures, discussion or debate formats for mini-cases during workshops and public hearing format for descriptive cases during assessments.

Based on the earlier discussion of desired skills of a software architect and a study by Clements, Kazman and Klein et al. [8] on the duties, skills and knowledge of software architects, the required elements (RE) of a case for software architecture involve:

RE1. Description of stakeholder and their requirements that require learners to investigate new architectural styles and the required components to support the style.

RE2. Description of business and technical environments, tools, technologies and other necessary components to enable the learner to describe an architecture in the form of architectural artifacts.

RE3. Description of constraints to enable the learners to reason their architectural decisions and justify the trade-offs among software qualities.

4 IMPLEMENTATIONS OF OUR CASE-BASED DESIGN FOR SOFTWARE ARCHITECTURE

In this section, we address RQ1.2 with a discussion of how we implement bullet-cases, mini-cases, and descriptive cases in lectures, workshops and assessments as shown in Table 1. We focus our cases in the typical architecture domains of the web, mobile, cloud and enterprise system architectures. We design these cases in a guided format for learners to be able to follow the steps to address specific issues identified in these cases.

4.1 Implementing Bullet-Cases in Lectures

Bullet-cases are good to discuss during a lecture for students to reflect on an important concept. We typically inject bullet cases after about 15-30 minutes of slides, which provide a good change in teaching style from passive slide-based lectures to discussion-based. Learners are typically grouped into small groups of 2-3 or individually to address the case questions.

As an example, after a lecture on ISO 25010 software qualities [9], which can be dry given the long list of software qualities, we introduce a bullet case where specific examples (e.g. online banking when one-time authentications or OTP are required) are discussed for the potential quality trade-offs and mitigation actions to minimize the trade-off. Using the same example of online banking, we highlight that the design for confidentiality trade-offs the usability of the system and can be mitigated if OTP is required only at certain transactions instead of at the login page. Students are also welcomed to give their own examples that they have experienced.

4.2 Implementing Mini-Cases in Workshops

For mini-cases, learners form larger groups of 4-5 and we use both the discussion and debate format. The learners are required to solve specific questions in these practical business cases, present and discuss with the class. The duration of the workshop can take between 60 minutes to 2 hours. Each group might be doing the same case or a different one. They are playing the role of an architect and have to address the stakeholders' concerns, played by other groups of learners in the class. We conduct a reflection session after the learners have presented which is vital for the learners to recap on their understandings and for instructors to ensure the learning outcomes are achieved.

An example of a case is a web/ mobile enterprise system integrated with some internet of things (IoT) devices that are monitoring the environment, which provides continuous environment updates such as haze and weather information. The learner's role is to design this enterprise system with efficient

delivery of this information to the users in a timely manner. A point to ponder is the scalability of the design that is required to be handled situations when a large number of concurrent users are assessing the system during serious environmental conditions. We discuss the usage of caching instead of retrieving the data from the internal system to increase performance. Another case adopted is about a multi-tenant system to be built and deployed on the cloud. The learner's role is to review the requirements and designs on how to handle the multi-tenancy aspect of the system. There are design decisions and trade-offs for each of the multi-tenancy options that impact qualities. Points to ponder include the decisions of shared vs isolated designs. Deciding on a shared platform might ensure a lower cost but trade-off the security aspect. Deciding on a no-shared platform isolates the system in terms of security but at a more significant cost and maintenance for the service provider [10].

4.3 Implementing Descriptive Cases in Assessments

For descriptive cases used in assessments, learners form groups of 4-6 and we use the public hearing format. The learners are exposed to case studies that are of a broader scope and require them to make more decisions and produce more concrete architecture deliverables as in a real-life project. Each group analyse the case and provide their recommendations with justifications to the rest of the groups within 3-5 class sessions. Other groups are allowed to participate with questions during the public hearing session.

An example of a descriptive case being adopted in our course is when the learners are given a real-life integration case study, and they are required to address specific key issues on enterprise integration. In this case study, the company wishes to migrate its legacy mainframe system to a modern application architecture based on Java. During the migration phases, they wish to adopt an architecture also to migrate the subsystems integrated into the legacy system. The learner's role is to design the integration architecture for the migration of the subsystems. The learners are given the current architecture diagram and interface details of each subsystem and the legacy system with specific requirements such as minimum disruption to the subsystems. Points to ponder include the performance impact due to the additional layer of computation between the legacy system and subsystems when the learner designs an enterprise service bus (ESB) between the systems.

In another example of a descriptive case, we describe a case of a nationwide healthcare system designed with the objective to monitor students' health in primary, secondary and tertiary institutions. The learner is required to architect a distributed system that still addresses the security, performance, maintainability and scalability qualities. Decisions made on the architecture design can favour one of the qualities but likely trade-off another. For example, the learner will need to decide whether to first persist the healthcare data in the storage available in each institution during the screening process or directly access a centralised remote system to persist the data. Adopting the former can allow for better decoupling of the system but risk data inconsistency across all institutions. On the other hand, adopting the latter can achieve better maintainability of the architecture and data consistency but risks single point of failure (SPOF) at each institution. For each trade-off, the learners should be able to recommend mitigation actions. In the earlier case, the learners can propose redundancy designs to address SPOF.

We also change the style in certain classes to allow the teams to propose their own case. In this case, instructors evaluate these cases and inject specific questions to be addressed instead. This format helps the learners to be exposed to more variety of case studies but increase the difficulty for the instructors to have a standard rubric to assess their performance. We partially addressed this concern by incorporating feedbacks from other teams in the final assessments.

5 Evaluation of our Case-Based Design for Software Architecture

In this section, we seek to address our second research question RQ2. We break down RQ2 into 2 specific questions.

RQ2.1 Is our case-based design effective for our learners to learn software architecture?

RQ2.2 What are the factors in our learners' profile that impact the effectiveness of our case-based design?

We first describe our evaluation design, followed by an evaluation analysis to address RQ2.1 and RQ2.2. The evaluation of our case-based design is conducted over 9 class runs in 8 years for 386 postgraduate students taking a software architecture course with our proposed case-based design.

5.1 Evaluation Design

The learners are required to provide survey ratings based on a 5-point Likert scale to the following key survey questions:

(1) "Is the case-based sessions effective to learn software architecture?" with ratings of "1-Poor, 2-Unsatisfactory, 3-Satisfactory, 4-Good, 5-Excellent."

(2) "What is your expected grade for the case-based assessment?" with ratings of "1-F, 2-D, 3-C, 4-B, 5-A".

The survey ratings and information of the learners' profile are averaged out over the number of students in each course run. Table 2 shows these average ratings obtained from the first class in 2010 to the last class in 2017. There are two class runs in 2010 and one class run for the rest of the years.

We compile their profile based on their postgraduate admission data in terms of their years of working experiences, their basic degrees and their job domain. We categorised their basic degrees into 3 types - (1) IT-Related (e.g. Computer Science, Information Systems, and Information Technology), (2) Engineering (e.g. Electrical, Electronics Engineering) and (3) Non-IT/Engineering (e.g. Mathematics, Physics). In the column "Proportion of Learners with IT-Related Basic Degree" of Table 2, we show the percentage

Table 1. Implementation of Proposed Case-Based Design and Implementation

Case Type and Objective	Case Format	Case Conduct	Case Duration	Examples of a Case and its relationships to the required Case Required Elements
Bullet-Cases - Reinforce single teaching point	Discussion Format	Lectures	15-30 minutes	Discussion on ISO 25010 software qualities (RE3)
Mini-Cases – Apply focused concepts in a practical scenario	Discussion and Debate Formats	Workshops	60 minutes to 2 hours	Web / Mobile Architecture for a Government Agency (RE2, RE3) Multi-Tenant Cloud Architecture based on e-Commerce (RE2, RE3)
Descriptive Cases – Detailed scenarios to apply multiple concepts	Public Hearing Format	Assessments	3-5 class sessions	Distributed Architecture based on a Healthcare domain (RE1, RE2, RE3) Enterprise Integration Architecture based on Insurance domain (RE1, RE2, RE3)

of learners with a basic degree that is of IT-Related type per class run. We categorised the learner's job domain into 5 types - (1) Academic and Research, (2) Financial and Insurance, (3) Government and Medical, (4) Services and Product and (5) Others (Telecoms, Transport, Energy). Our workshop and assessment cases scenarios are based on domains of type (2) and (3). In the column "Proportion of Learners with Case-Related Job Domain" of Table 2, we show the percentage of learners with their job domains that match the domains in our case scenarios per class run.

6 Evaluation Analysis

We first explain the general trends of the survey results to address RQ2.1, followed by an analysis of these results to address RQ2.2. For RQ2.1, the effectiveness ratings for the case-based design are generally good and above (above 4) except for two years falling just below the good rating. These ratings are considered good in the 5-point scale for our department, giving us confidence that the learners are able to learn software architecture using the case-based design effectively. The expected grade rating is fairly consistent over the years ranging between 4 and 4.4 except for the year 2012 with 3.974 which is still very close to 4 (B grade). For improvements, we seek to find out the reasons for the drop of effectiveness in 2013 and 2017 by performing a correlation analysis (as shown in Table 3) between the survey ratings and the learner's profile to derive more insights to address RQ2.2. Following are the key takeaways:

- We realised that the learner's years of experiences are highly negatively co-related (-0.815) to the effectiveness of the case-based design. A regression analysis of the learners' experiences against the effectiveness also shows a statistically significant relationship with a p-value of 0.007 (<0.01). This result implies that with longer the years of working experiences, the effectiveness of our case-based design actually decreases. A possible qualitative explanation is due to the guided-inquiry aspect of our case design. For learners' with less working experiences, the case design with guided-inquiry discussions enable these learners to understand and apply the knowledge gained. However for learners' with longer years of working experiences, the case design with guided-inquiry discussion might not be as effective as these learners might already being exposed to similar case scenarios which limit what they can learn from these cases. A possible suggestion based on the study [4] is to adopt another teaching approach - Problem Based Learning with open-inquiry.
- The effectiveness of the case-based design is also negatively co-related (-0.721) with the total number of students. A regression analysis of the total number of students against the effectiveness also shows a statistically significant relationship with a p-value of 0.028 (<0.05). It might be good to hear views from diverse groups of students, but it can be ineffective when the number of students is too large, and some students are no longer involved in the discussions. One possible mitigation strategy is to assign each group to understand at least one other group and they are required to pose challenging questions to that group.
- The proportion of learners with IT-related basic degree is positively related (0.66) to the effectiveness of the case-based design. Although we expect this outcome, the co-relations are not as strong as expected with the regression analysis showing a p-value of 0.051 (>0.05). We also realised that many of these learners took other diplomas and certifications, which might have made their basic degree less relevant to the effectiveness of our case-based design.
- The proportion of learners with case-related job domain does not show significant co-relations (0.066) to the average effectiveness rating. Even though our cases primarily focus on domains of Financial and Insurance, Government and Medical, the decision to focus on these domains does not impact the effectiveness of the case-based design to learn software architecture. We believe the reason is that the learnings of our cases are to a certain extent generic enough to be applicable to many other domains.

Table 2. Survey ratings and Learners' Profile

Year/Run	Survey Ratings		Learners' Profile			
	Average Effectiveness Rating	Average Expected Grade	Years of Working Experience	Total Number of Learners	Proportion of Learners with IT-Related Basic Degree	Proportion of Learners with Case-Related Job Domain
2010 Run 1	4.265	4.317	5.47	34	73.33%	28.57%
2010 Run 2	4.244	4.333	4.36	41	66.67%	32.00%
2011 Run 3	4.212	3.974	5.30	41	68.75%	19.05%
2012 Run 4	4.255	4.304	5.45	47	77.27%	22.50%
2013 Run 5	3.977	4.386	6.26	44	54.76%	14.29%
2014 Run 6	4.27	4.243	4.80	37	61.54%	14.29%
2015 Run 7	4.152	4.25	5.86	46	54.17%	16.67%
2016 Run 8	4.14	4.43	5.86	37	68.00%	6.90%
2017 Run 9	3.9	4.4	6.69	59	57.50%	28.57%

Table 3. Relationships between Survey Ratings and Learners' Profile

	Average Effectiveness Rating	Average Expected Grade
Years of Working Experience	-0.815	0.199
Total Number of Learners	-0.721	0.191
Proportion of Learners with IT-Related Basic Degree	0.666	-0.169
Proportion of Learners with Case-Related Job Domain	0.066	0.004

- The average expected grade rating is not showing any significant co-relations to any of the factors in the learners' profile. A possible reason is that the expected grade is more related to their type of skills rather than the years of working experiences or basic degree or job domain. We realised that learners with prior job scope requiring them to make architectural design and decisions tends to be more confident of their performance when taking this course.

7 Threats to Validity

The learners' years of experiences and job domain obtained during their postgraduate admission might differ when they actually take the course. They can take this course in their first or second year and we adjust their years of experiences accordingly. Their job domain might also differ and we further verify and adjust accordingly against their LinkedIn profile if available.

Our findings are on the application of CBL with the guided-inquiry format in a postgraduate software architecture course. These findings remain to be validated in other course scenarios (e.g. software design, software testing) or other groups of learners (e.g. undergraduates) or other formats (e.g. open-inquiry).

8 Conclusion

Teaching software architecture to postgraduate students poses challenges due to its inherent level of abstraction. In this paper, we describe how we design case-based learning into our course to enable our postgraduate students to learn software architecture more effectively. This paper explains the benefits of applying case-based learning that corresponds well with the required skills of a software architect. We provide how we design our cases for software architecture and how we implement these cases in a postgraduate course over a period of 8 years.

We evaluate the effectiveness of our case-based design in software architecture based on survey ratings and analyse the relationships of the survey ratings to our learners' profile. Our survey ratings show that our proposed case-based design in software architecture is effective for our group of learners. Our analysis results show the learners' years of working experiences and the number of students have significant negative co-relationships to the effectiveness of the case-based design. On the other hand, the learners' job domain and basic degree do not have significant co-relationships with the effectiveness of the case-based design.

We hope that these findings can guide course designers to design case-based learning for courses with similar challenges and better understand the relationships of the learner's profile to the application of case-based learning. In our future work, we seek to compare and analysis our current case-based learning design with other design approaches and investigate across groups of adult learners including undergraduate and more experienced working adults to derive new insights.

REFERENCES

[1] M. Galster and S. Angelov, "What makes teaching software architecture difficult?" In IEEE/ACM International Conference on Software Engineering Companion (ICSE-C), 2016.

[2] R. C. d. Boer, R. Farenhorst and H. v. Vliet, "A community of learners approach to software architecture education." In 22nd Conference on Software Engineering Education and Training, 2009. CSEET '09, 2009.

[3] M. Srinivasan, M. Wilkes, F. Stevenson, T. Nguyen and S. Slavin, "Comparing problem-based learning with case-based learning: effects of a major curricular shift at two institutions." In Academic Medicine, 82(1), 74-82, 2007.

[4] E. L. Ouh and Y. Irawan, "Teaching Adult Learners on Software Architectural Thinking Skills." In Frontiers in Education, 2018.

[5] C. R. Rupakheti and S. V. Chenoweth, "Teaching software architecture to undergraduate students: an experience report." In IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), 2015.

[6] E. L. Ouh and Y. Irawan, "Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course." In Frontiers in Education, 2018.

[7] V. Saini, P. Singh and A. Sureka, "Seabed:An open-source software engineering case-based learning database." In Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual (Vol. 1, pp. 426-431). IEEE.

[8] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy and P. Verma, "The duties, skills, and knowledge of software architects." In Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on (pp. 20-20). IEEE.

[9] "ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models." 2011. [Online]. Available: https://www.iso.org/standard/35733.html.

[10] E. L. Ouh and S. Jarzabek, "An Adaptability-Driven Model and Tool for Analysis of Service Profitability." In International Conference on Advanced Information Systems Engineering. Springer, 2016.

.