10-2018

Teaching adult learners on software architecture design skills

Eng Lieh OUH
Singapore Management University, elouh@smu.edu.sg

Yunghans IRAWAN

Teaching Adult Learners on Software Architecture Design Skills

Ouh Eng Lieh
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 architectures present high-level views of systems, enabling developers to abstract away the unnecessary details and focus on the overall big picture. Designing a software architecture is an essential skill in software engineering and adult learners are seeking this skill to further progress in their career. With the technology revolution and advancements in this rapidly changing world, the proportion of adult learners attending courses for continuing education are increasing. Their learning objectives are no longer to obtain good grades but the practical skills to enable them to perform better in their work and advance in their career. Teaching software architecture to upskill these adult learners requires contending with the problem of how to express this level of abstraction practically and also make the learning realistic. We provide here our seven years' history of teaching software architecture of information systems to these adult learners as a part of a public continued education program. We describe our key challenges encountered at these levels and go on to analyze the results of the course design we have taken over the years. We also compare our teaching methods based on problem-based and case-based learning and examine their effectiveness for adult learners. We hope this discussion can help educators design and improve software architecture curriculum and support education researchers in investigating pedagogical approaches and tools to better support adult learners.**

Keywords—software architecture, adult learners, case-based learning, problem-based learning, peer learning

I. INTRODUCTION

Software architecture remains an abstract and challenging subject for learners to grasp and for educators to teach. In most cases, a system that can be comprehended quickly by learners would be too simple to have a significant architectural discussion. The skill set expected for a software architect is also multi-faceted which increases the level of difficulty to train one to be a competent software architect. The role of a software architect typically entails one to have the technical skills that minimally include software design and development experiences and understanding of system and networking concepts, analytical skills to grasp the problem quickly, diagnose the possible root causes and leadership skills to make significant decisions for the project based on the context and environment.

With the rapid advancements in the technology world, the proportion of adult learners attending information technology courses are increasing. Adult learners typically bring with them a wealth of practical experiences and in some cases, also constraints within their field or organization. Their key learning objective is to learn the practical skills to for them to perform better in their work and advance in their career. Concepts and theories are great to know but if the learners do not find them practical in the real world, they will not be able to apply it. Our findings are derived from teaching software architecture of information systems to adult learners who are information technology (IT) practitioners having a certain degree of IT experiences and wish to or require to upskill in software architecture design.

One key challenge is to be able to impart the *practical skills* such that the adult learners can apply immediately on their information system projects when they are back to work. From an educator's perspective, we need to recognize the importance of imparting practical software architecture skills. We should seek to validate the practicality and relevance of the course contents by asking questions such as "Is this methodology too heavy-weight?", "Does this model which measure a specific software quality requires the components to be built first?", "When should I use this architectural style?" and "When should I not use this architectural style?" For example, when given a problem of enterprise application integration, the learners tend to adopt the broker style using the enterprise service bus (ESB) to transform and route messages to achieve loose coupling between software systems. Although there are many useful features in a broker, implementation of a broker can also be impractical due to organizational constraints. An organization with an existing tightly coupled integration architecture but with only a handful of software systems might not see the need to re-design their architecture with a broker and undertake project risks and additional costs. The cost of implementing an ESB can also outweighs the benefits to the organization. The learners must be aware of such considerations in order to apply these skills effectively in their workplace. During the conduct of the course, we usually find ourselves either giving or being asked by learners to give examples of how this concept, software, model or tool can or had been applied in an information system project.

Another challenge is to be able to design the course that is *relevant to the learner's environment*. Software architecture is defined as the fundamental organization 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] [2]. The learners' environment can differ from each other which can influence the significant decisions to be made to the software architecture. For example, a software

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

architecture which proposes to implement Simple Object Access Protocol (SOAP) web service call to integrate two systems might work in most environments but will not work if the two systems are located on opposite side of an "air gap" network environment. While the components and their relationships in the software architecture can be taught using a lecture-based method, the environment and principles aspects vary with the learner's experiences and background. It is not possible to completely cover the variants of the learner's environment given the limited course duration.

Another challenge in teaching software architecture to these adult learners is how to make these abstract concepts *easy to comprehend*. To learners new to architectural design, this is a mindset switch from the concrete aspect of software programming and low-level software design to abstract architectural design. 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 any concrete output and there is no one single perfect solution is a frustrating divergence from what they are doing.

When we design our approaches for the course, we need to consider these challenges in totality and not as separate challenges. Failure to do that can disengage the learner's interest and affects their learning outcomes. For example, we initially thought by demonstrating program codes that can achieve a specific software quality, it can help them to better comprehend abstract architectural design concepts in a practical manner. However, this approach surprisingly backfires several times due to their prior IT experiences of the learner and also due to the longer time taken to teach and learn. More experienced learners commented that this is too low level and not necessary while less experienced learners got distracted by the programming aspect. During the design and conduct of the course, it is crucial to be aware of the adult learners' background, pace the lesson accordingly and find the right balance.

Our pedagogical approach to deliver the software architecture course centered on applying a combination of traditional lecture-based and workshops. The lecture-based session focuses on imparting the underpinning knowledge in architecting a software system and providing the essential background for the learners to do the workshops. For the conduct of the workshops, we have two variants– one based on the problem based learning (PBL) method and the other based on the case-based learning (CBL) method. These methods are already widely applied in domains such as the medical fields. For workshops based on PBL, we group the learners into teams of 4-5 and they are given an open-ended problem with many exit solutions. Instructors play a minimal role and do not interfere in the discussion, even when learners explore off the tangents. For workshops based on CBL, we also group the learners into teams of 4-5 and give them a case-study with some advance preparation. When learners begin to explore off the tangents, instructors can play as facilitators and use guiding questions to bring them back to the primary learning objective. In this case, both students and instructors share the responsibility to draw conclusions on the learning points. Both variants do have something in common which is to allow the learners to focus on the process of discovery, stimulate problem-solving, independent learning, and teamwork. Some have called PBL an open inquiry approach while CBL is a guided inquiry approach. We evaluated the impact of these two methods to the adult learners when they give their ratings on relevance and practicality in the course feedbacks.

We seek to address the following two research questions for the rest of the paper.

1. How do we design the software architecture course for adult learners that is practical, relevant and easy to comprehend?
2. Which teaching method (PBL or CBL) enables the adult learners to learn more effectively in terms of practicality and relevance?

The rest of the paper is organized as follows: We first present the related work in Section II and the background of our course in Section III. Our course design and its evolution from 2011 to 2017 are discussed in Section IV to address our first research question. We describe how we apply PBL and CBL methods in Section V to address our second research question. The course feedbacks and analysis are summarized in Section VI to measure the effectiveness of our approaches to both research questions. The study results show that our efforts to design a course that is practical, relevant and easy comprehend are rewarded with improving feedback ratings from our profile of adult learners who also express higher preferences for the teaching method based on PBL. Threats to the validity of our result are discussed in Section VII and the conclusion in Section VIII. The two key contributions in this paper are: (1) the design of a software architecture course for adult learners and (2) Insights into the pedagogical approach adopted for adult learners.

II. RELATED WORK

A framework to classify software architecture teaching challenges: Galster and Angelov [3] describe the framework involving the relationship of concept (software architecture), representation (architecture description), referent (software architecture practice) to the learner element in the learning space. In addition to the vagueness of the concept of software architecture itself, architecture problems are usually "wicked". Asking students to create an architecture is different to e.g., asking them to write a Java program - students have a much clearer understanding of what the expected outcome is. Visualizing and document software architectures can be challenging and clear guidelines on what and how to document has to be given. There is often a distinction between high-level architecting and low-level implementation activities. Overcoming this requires the creation of a mindset in students to not treat architectural decisions in isolation.

The need to teach software architecture design course that is industrially relevant: Mannisto, Savolainen and Myllarniemi [3] discuss on the means for teaching students what it takes to face software architecture design problems with some characteristics of wicked problems and providing students some methodological tooling for coping with the problems in their profession as software architects. Industrial

environments can differ significantly from typical exercises in software architecture teaching and constraints often dominate the development process. Ouh and Jarzabek [4] demonstrate how constraints regarding tenant base or service costs affect the service architectures and eventually service profitability. We agree to the need of the course to be industrially relevant and our course for adult learners is designed with this in mind.

Challenges in teaching software architecture courses to undergraduates: Rupakheti and Chenoweth in [5] described their experiences and learnings in teaching software architecture course to undergraduates. Their systematic problem in getting architecture concepts across to undergraduates is similar to our challenge for learners, primarily those with programming but limited design experiences. They advocated that the overall design of their software architecture course at the undergraduate level should consider how ambitious the course developer's goals will be, given the students are unlikely to take the role of the architect in their first job. In terms of our course design for adult learners, we seek to be practical and relevant so that the learners can apply their new skills immediately in their work after the course.

A Community of Learners Approach to Software Architecture Education: Boer, Farenhorst and Vliet in [6] described their experiences on applying community of learner's approach in which students are treated as partners in the knowledge development process to learn about the wicked nature of software architecture. Wicked problems cannot be solved by following a strictly sequential and fixed number of steps and the inquiry learning cycle of the Community of Learners approach instead provides for a good fit with the wicked nature of software architecture design. We do find this mode of teaching useful especially for adult learners who are able to learn from other learners willing to share their project experiences.

Comparing Problem-Based Learning with Case-Based Learning: Effects of a Major Curricular Shift at Two Institutions: Srinivasan, Wilkes, Stevenson, Nguyen and Slavin in [7] described 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 centers. The results of their study showed that learners and faculty overwhelmingly preferred guided inquiry-based of CBL over open inquiry-based of PBL for their medical curriculum. In our case, the study results show otherwise. The possible reasons for this difference might be due to the learner's profile, experiences and needs.

In this section, we highlight some of the existing work and their challenges to design software architecture courses, primarily focused on undergraduates or university level students. We also highlight the comparison of the case-based learning and problem-based learning methods applied in the medical curriculum. This paper seeks to address the education of another unique group of learners (adult learners) for a software architecture course and assess the effectiveness of applying the PBL and CBL methods for this group of adult learners.

III. COURSE BACKGROUND

A. Adult Learners

Over seven years, we developed, delivered and evolved a software architecture course to meet the needs of the IT practitioners taking our public course as part of a nation-wide continued education program. The continued education program is a government initiative to incentivize IT practitioners to upgrade their skills in a series of accredited courses.

The profiles of the learners are typically IT practitioners having an average of 5 years working experiences, taking the roles of program project manager, project lead, senior software engineer and business analyst. There is a small percentage of outliers – learners with less working experiences but these learners generally have sufficient technical background.

B. Course Objectives

The primary objectives of the course during is to prepare adult learners to appreciate, design and evaluate software architecture designs in the industry. These are the expected learning outcomes which are also reviewed and accepted by the relevant authority of the continued education program to assess the learner's competency.

1. Explain how the application architecture fits into the broader context of organizational business goals and enterprise architecture.
2. Design the architecture with an emphasis on the common application integration components.
3. Describe the software architecture with views and viewpoints
4. Analyse software architecture designs with respect to the quality attributes and their tradeoffs.

C. Course Structure

The course schedule is structured around a 32.5 hours classroom contact time and can be delivered in consecutive days or in weeks, allowing us the flexibility in conducting for adult learners who are working IT practitioners. For example, when a company decides to enroll a substantial number of their staff for this course, we will have to conduct the lesson according to the availability of these company participants.

The course agenda is split into 2 broad categories of topics in architectural fundamentals and software qualities. For the topics on architectural fundamentals, learners will get to appreciate the different types of architectures styles and reference architectures. For the topics on software qualities, learners will get to learn key concepts in software qualities such as availability, security and performance to further improve the quality of their architecture. These topics can be conducted either purely lecture-based or with workshops. However, we ensure that the proportion of the workshops time is at least 50% of the total course duration for the learners to have the opportunities to work on problems based on what they have taught in the classroom. The structure and conduct of these workshops are discussed with more details in Section V.

At the start of each course, we conduct a kick-off session for the adult learners, primarily to understand the learner's background and needs. This session also helps the learner to better reflect on their learning outcomes at the end of the course.

IV. Course Design and Evolution

The design of an actual software architecture can be used as an analogy to design a software architecture course. The topics (components) have to be correctly integrated (relationships) and right course design approaches (principles) have to be made to accommodate the learner (environment). Our course design focus on architecting for information systems which most of our learners required back at their workplace. A summary of the following discussions and its applicability to address earlier stated challenges are available in Table I.

A. Coverage of Architectural Styles

Architectural styles like design patterns are the general reusable solution to a commonly occurring within a given context but at the architectural level. In the first few years, we cover these the traditional styles of client-server, data-centric, event-driven, and layered that are generally more applicable for an information system architecture. In the subsequent years, we structured them into structural styles (component), communication styles (relationships). Structural styles relate to how the components are structured in a software architecture while communication styles are the relationships among the components. We categories the common architectures that utilize these structural and communication styles into reference architectures. We regularly review our contents for their relevance to the industry practices before each run. For example, we included the reference architectures on Service-Oriented Architecture (SOA), Microservices, Broker, Cloud Computing, Internet of Things, and Big Data.

Over the years, the time allocated to the traditional styles gradually reduced to allow for more recent reference architectures. The conduct of the architectural patterns session also moved from explaining the "what and how to implement each style" to "What are the considerations when using this style?" We further provoke their thinking to understand the style deeper with this kind of discussion. This change is an example of how we keep the course practical and attempts to address the diverse background of the adult learners. Using examples and analogies to explain each of the styles to make it easier to comprehend. For example, receiving mobile app notifications is a good case of how publish-subscribe works.

B. Coverage of quality attributes

Bass, Clements, and Kazman in [8] described an approach to use quality attribute scenarios to document the qualities of availability, modifiability, performance, security, testability and usability. In our initial course design, we use the same format of the quality attribute scenarios but adapt for information systems which are more relevant to our learners. Over the years, we also adopted the format of the misuse cases due to its relevance to other artifacts in the Rational Unified Process (RUP) [9] that is already widely adopted by our learners. However, we do acknowledge that the quality attribute scenarios are more structured and specific to describe qualities scenarios.

During 2013, the coverage of the quality attributes is evolved to refer the list of software qualities in ISO 25010 Product Quality Model [10] which provides a more comprehensive and clearly defined set of quality attributes. Using the ISO standard gives learners a commonly accepted list of software qualities to discuss and apply.

The lesson conduct for quality attributes evolved from understanding the quality attributes and describing the quality attribute scenarios to including trade-offs and mitigation measures, increasing the practicality of the lesson. For example, one question we typically asked is whether we should separate web, application and database tiers into multiple nodes with firewalls in between to limit access and monitor data for confidentiality (security). This design improves security but introduces a tradeoff which is the additional points of failures (availability). The availability tradeoff can be mitigated to a certain extent by clustering the nodes for higher availability. Another example is the development of a product-line improves maintainability but tradeoff additional upfront costs which can be mitigated with an adaptive reuse technique as described by Khue, Ouh and Stan [11] for a set of related mobile apps and by Koznov, Luciv, Basit, Ouh and Smirnov [12] for a set of related software technical documentation. Due to the learners' project experiences, the outcomes of the described scenarios are mostly unique. To make it relevant to their environment, we design for the learners to describe their real-life design decisions and we discuss the potential software qualities, trade-offs and how they can mitigate the trade-offs. This kind of discussion becomes an interesting learning process for both the learners and the instructors and it is also a good exercise for the learners to switch their mindset from software design and development to architectural design. We also tend to give thought-provoking questions for learners to further reflect on their learnings. For example, we ask "Does designing for security always negatively trade-off other qualities in your architecture? We often find it is useful and important to highlight the fundamental theorem of software engineering - We can solve any problem by introducing an extra level of indirection [13] with the extension "except for the problem of too many levels of indirection.", emphasis that each design decision also has its own impact and trade-offs.

C. Practice of soft skills

Soft skills such as communication and leadership skills are essential for an architect to lead, present, negotiate and justify their architectural designs and decisions. In this course, the learners are given the opportunities to take turns to lead their team and present their designs in the workshops.

We generally do not have many issues with their presentation skills as the learners have been practicing their soft skills in their workplace. However, we do find the lack of a solid technical foundation in some learners when they justified their design decisions which constrained their ability to explain their designs convincingly.

TABLE I. SUMMARY OF THE COURSE DESIGN AND EVOLUTION

Dimension and Applicability - (1) Practical Skills (2) Relevance (3) Easy to Comprehend	Initial Design	Current Design
Coverage of Architectural Styles		
Type of Architectural Styles (2)	Client-server, tiered computing data-centric, call-and-return, event-driven layered, virtualization	Added reference architectures on SOA/Microservices, Cloud Computing, Internet of Things, Big Data
Lesson Conduct (1) (2) (3)	Focus on "what and how to implement each style."	Focus on "What are the considerations when using each style?" Use more examples and analogies.
Coverage of Quality Attributes		
List of Quality Attributes (3)	Availability, modifiability, performance, security, testability and usability	List of software qualities in ISO 25010 Product Quality Model.
Description of Quality Attributes (2)	Quality attributes scenarios with examples focused on information systems	Inclusion of misuse case.
Learning Conduct (1) (2) (3)	Understanding the quality attributes and describing the quality attribute scenarios	Including discussion on the trade-offs and mitigation measures
Practice of Soft Skills		
Leadership skills (1) (2)	Workshops Project presentation	No Change
Communication and Negotiation Skills (1) (2)	Justifications of decisions made including technical explanation and trade-offs	No Change
Usage of Tools		
Types of Tools (2)	Commercial Tooling	UML Modeling, Free Form – paper and pen
Purpose of Usage (1)	Documentation and View Consistency	Notation Consistency
Design Method and Assessment		
Selection of Method (1) (2)	Rational Unified Process (RUP) The Open Group Architecture Framework (TOGAF)	No Change
Design of the Assessment (1) (2)	Based on written deliverables (70%) and participation in workshops (30%)	We scale down the assessment portion of written deliverables (50%) and focus more on their participation in workshops (50%)

D. Usage of tools

The usage of the tools to achieve the learning outcomes has an interesting development. Initially, we would like the learner to learn a new tool that allows them to document their designs and at the same time, check for inconsistencies in their views. We introduce a commercial tool to do that and the learners have to install, learn to use the tool (if they have not) and address the questions in the required deliverables. However, we realize the use of learners' time is not maximized within the limited course duration. Subsequent feedbacks are also mixed with more than 50% indicated that they would rather have a simpler UML Modelling tool or even paper and pen due to the impracticality of the commercial tool to be applied immediately back at their workplace. Over the years, we decide to fall back to the key objective of drawing an architectural view which is to effectively communicate the design that addresses stakeholder's concerns. We confine ourselves to only teach the notations and its purpose in the architectural view, usage of our proposed tool or any other tools (e.g., paper and pen) are acceptable as long as the notations are articulated clearly and effectively.

E. Design method and Assessment

The selection of the right methodology is being debated during the initial course design. There are many available architecture design methodologies that can be adapted for a software architecture course. The criteria lie in the usefulness of the method to the learner's workplace and the reputation of the method in the industry. We narrowed down to Rational Unified Process (RUP) [9] and The Open Group Architecture Framework (TOGAF) [14]. Eventually, we adopted RUP primarily due to the existing adoption of the same method in other courses in our institution including one of our postgraduate programme. Both of these methods are also accredited under the Open Group Certified Architect (Open CA) program [15] which allows the learners to apply the methods in their workplace and subsequently submit their experiences for Open CA certification. This decision stays since the beginning of our course design.

The goal of the assessments is to determine the level of competency of the learner. The adult learners need to achieve a certain level of competency to enjoy the incentives provided by the relevant authority for taking this course under the continued

education program. We evaluate the learners based on their written deliverables to the workshops and observe their participation in workshops as to whether they sufficiently demonstrate their capabilities to practically apply the concepts and methods. For passive learners in the workshops, we find opportunities for them to participate by consciously rotating the learner's turn to present. As these adult learners are being assessed for their competency to apply the knowledge gained in the course, they are not given a specific grade but a general "competent/not competent" result. Their real test is applying these learnings back at their workplace, not in the classroom. Due to this reason, we scale down the number of written deliverables being assessed and focus more on their active participation in workshops over the years.

This section addresses the first research question on how we design our software architecture course for adult learners. We evaluate the effectiveness of our course designs decisions made over the years when we analyze the course feedbacks in section VI.

V. CONDUCT OF THE WORKSHOPS

For the learners to better appreciate the software architecture and the practical aspect of it, we conduct workshops during the course duration, applying the problem-based learning and case-based learning methods. A summary of how the workshops are conducted is given in Table II.

For problem-based learning, we assign open-ended architectural problems to each group. Within their group, learners are supposed to discuss their design solution, research on more information and present their findings and conclusion to the instructor and other learners. During the conduct of the workshop, learners play the role of an architect and the instructor play the role of a facilitator providing minimum guidance in the architecture design.

For case-based learning, each group is given some readings and articles on a project scenario which they are required to read before the lesson. Each group is required to solve the given questions within the scenario. The scenario involves architecting a complex distributed information system and the learners to apply the architectural concepts and address challenging software quality requirements. During the conduct of the workshop, learners play the role of an architect and the instructor play the role of the other stakeholders (for example, healthcare staff, internal IT team) clarifying functional requirements with guidance to the technical design of the architecture.

For both methods, we adopted a peer learning strategy for each group to hear comments from other groups. However, there is a minor problem. Constrained by the presentation time, the learners might not be able to grasp the problem or the design solution of another group thoroughly to evaluate it and limits their capability to ask questions during the presentation. We modified the conduct of the workshop by dividing the learners of each group into two sub-groups. One of the sub-group is required to explain their design solution. The other sub-group is required to understand the design solution from another assigned group so that they can prepare their questions for the actual presentation. This *peer-learning* greatly advances the learning process as now each learner will be able to hear from multiple learner's experiences and thinking, applied to a real-life scenario.

VI. COURSE FEEDBACKS AND ANALYSIS

A. Course Runs

The public course is conducted 4 times a year with an average of 15 learners per class in the earlier years to currently 25 learners per class in recent years. Each run of the class spans over 5 full days and conducted by two instructors to give the learners additional exposure to the experiences of different instructors. We have trained 524 IT professionals from the beginning of 2011 to the end of 2017 over a total of 28 runs of the course. There is no cancellation of the scheduled classes since the launch of the course.

Each learner is given a list of feedback questions and they are required to hand in their answers and comments at the end of the course. The metric-based questions are rated on a 5-point Likert scale metric ranging from 1 – strongly dissatisfied to 5 – strongly satisfied.

The four questions asked for the overall of the course are: (1) "Does this course fulfilled course learning objectives and outcome?" (2) "How well does this course impart knowledge and skills needed for you to apply and practice?" (3) "How are the logistics and administrations provided?" and (4) "What is the overall satisfaction level for this course?" There are also questions for the adult learners to give feedback on the specific workshops. The four questions asked for the workshops are: (1) "Is the module conducted practical and applicable to your work?" (2) "How is the quality of materials provided?" (3) "Does it meets the module objectives?" and (4) "How is the level of class interaction?" The ratings for the course overall are available for every runs from 2011 to 2017. However, the ratings for each of the lecture and workshop session are only available for the runs from 2011 to 2016 due to a simplification of the feedback form that took effect in 2017.

There are also open-ended questions which allow the learners to write their comments on other areas of the course. The questions asked include "How do you apply the lesson learnt in the workplace?", "What do you like BEST about the course?", "What do you like LEAST about the course?" and "Do you have suggestions to improve the course?"

B. Analysis of the Public Course Feedback

For the evaluation of the course design to answer the first research question, we measure the ratings for the following metric-based questions - "How well does this course impart knowledge and skills needed for you to apply and practice?" and "What is the overall satisfaction for this course?" The rating results of these two feedback questions for conducting the course is shown in Fig. 1. It illustrates an upward trend in overall satisfaction and practicality of the course as rated by the public course participants over time, indicating that the improvement to the course design over time has effectively improved the public course learners experience in going through the course. Using Analysis ToolPak Add-in for Microsoft Excel [16], we perform a correlation analysis

TABLE II. CONDUCT OF THE WORKSHOPS

Instructional Element	Workshop Example based on Problem Based Learning (PBL) method	Workshop Example based on Case-Based Learning (CBL) method
Problem / Case Study	Learners are required to address a problem – How to support the existing systems to integrate to a new backend system.	Learners are given a School Healthcare Project scenario and they are given specific questions to be addressed during the design of the architecture
Time allocated for each session	2 hours	4 hours
Learner Pre-Preparation	None.	Pre-readings related to the School Healthcare Project such as requirements and background are given before the class.
Instructor Pre-Preparation	Effort to design the problems ensuring limited but sufficient information are given in the class to allow the learners to have open-ended discussions	Effort to design the workshop case study scenario and questions to be answered
Instructor's approach during the session	Act as the facilitator providing minimum guidance but can give alternative views to provoke discussions.	Act as the stakeholder of the project, clarify the requirements and give guidance in the actual design of the architecture.
(CBL) Questions to be addressed (PBL) Comments to provoke discussions	"Will the existing systems be adversely impacted by your architectural decisions?" "Is enterprise service bus (ESB) required for this project?" "Does introduce another layer of ESB degrades performance?"	"What are your logical and physical designs of the architecture?" "What are your measures to ensure availability and security of the system in the school?" "What do you think are the quality trade-offs in your design and how can you mitigate these tradeoffs?"
Assessment Method	Adopt a peer learning strategy where the learner is able to hear from multiple learner's experiences and thinking	
Learner's Deliverables	Presentation Self-Reflection Report	Presentation Self-Reflection Report Software Architecture Report

between this two set of ratings and the resulting value is 0.84, indicating a high correlation between the practicality and applicability of this course and its relationship to the overall satisfaction.

We also review the textual comments from the open-ended questions to understand with regards to the course overall and the perform sentiment analysis based on Amazon Comprehend service [17]. This service returns a score between 0 – 1 indicating the degree of confidence for that sentiment. The positive responses with a positive sentiment score higher than 0.75 generally fall under the following categories:

- Participants can understand the abstract concepts taught easily with practical examples. E.g., "Both instructors are very knowledgeable. One provided a lot of actual use cases to relate to the topic on hand, while the other uses appropriate analogies to explain the concept easily."
- Appreciation of abstract architectural thinking. E.g., "I learnt to think like an architect as in how to solve and plan problems in a higher view, the strategies and principles will definitely help to solve security, performance and maintainability in my current project."

The negative responses with a negative sentiment score higher than 0.75 gathered from the participants also have some common themes:

- The course duration is too short to cover the all the concepts. E.g., "5 days was too rushed for the course".
- Participants are not familiar with the type of system used in the examples. E.g., "Too much information to digest for me because I have the least background on web application."

The negative sentiments illustrate the constant challenge that we always have in terms of balancing the amount of time with the amount of teaching materials that we wish to cover. The diversity of the past experiences of the course participant also post a challenge of making some examples of a particular domain to be less relevant to other participants. We seek to provide some fundamental background knowledge to our course contents as pre-reading materials so that participants can be more prepared for the course.

For the second research question on our teaching methods in workshops, we evaluate the ratings for the feedback question "Is the module conducted practical and applicable to your work?" for the workshops conducted based on PBL and CBL. Fig. 2 shows the overview of the trend based on this feedback to this question, illustrating that ratings for the PBL being consistently higher than CBL. Using Analysis ToolPak Add-in for Microsoft Excel [16], a paired sample t-test for the workshops ratings based on PBL and CBL indicates a significant difference in the ratings for PBL (mean=4.12, variance=0.03) and the ratings for CBL (mean=3.96, variance=0.04) with $t(21)=4.54$ and p-value=0.0002.

Some learners commented that they could discover and learn more because of the open inquiry aspect of the workshop based on the PBL method and workshops based on the CBL method are not time efficient. Adult learners who need to sacrifice their working time to attend courses would also better

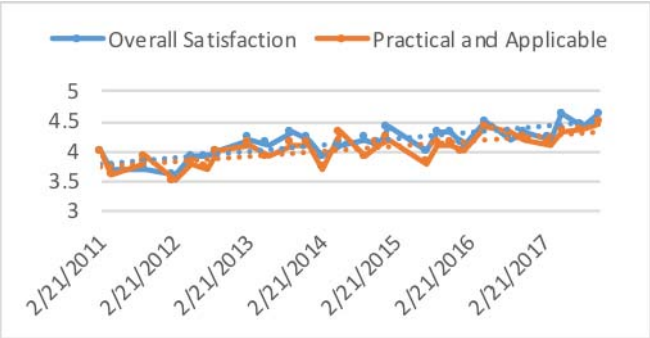

FIG. 1. OVERALL SATISFACTION RATING OF SHORT COURSE PARTICIPANTS FROM 2011-2017

FIG. 2. RATING FOR THE SESSION CONDUCTED BASED ON CBL AND PBL FROM 2011-2016

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. We suspect this difference may be due to the level of experiences of the adult learners and leave this be concluded in future studies. We will continue to fine-tune the course but we do not intend to completely replace the workshops based on the PBL method. We believe both methods address the needs of the learners to a certain degree depending on their profile and adds variance in our teaching methods. The structured characteristic of the CBL method also enables the level of competency to be assessed easily for the continued education program.

We are encouraged to see that our efforts to teach architectural design in a more practical and relevant manner is well-recognized by the participants over the years. Our use of case studies, analogies, demonstrations and discussions help in fostering a learning environment that learners can easily comprehend especially crucial in this today's fast-paced environment where abundant information is available over the Internet. However, we do caution to adopt this approach for less experienced learners as they might not be equipped with the necessary background knowledge to fully appreciate the lectures and workshops. From the instructor's perspective, substantial effort is required to prepare the course especially the workshops. There is also a need to consistently collaborate with the industry to stay relevant and practical in the course design.

VII. THREATS TO VALIDITY

While the software architecture has to be designed within the context of its environment, the findings in this study are also confined by the profile of our learners. Although we have accumulated these results over many years and our learners' profile varies in the type of industry and country, these results will require further validation when the environment (learner's profile) changes significantly.

The course contents are designed for learners who are required to develop software architectures of information systems. Even though these architectural design concepts are also applicable in other types of system (e.g., embedded systems), the design of our approaches might not be applicable (or have to be adapted) to a learner who is working on other types of systems.

VIII. CONCLUSION

We believe that with technology revolution and advancements in this rapidly changing IT industry, the need to learn software architectures become more evident as software architectures enable one to visually communicate the key features of each new technology in terms of the components, its relationship and principles of the architecture. However, teaching software architecture to adult learners poses unique challenges as compared to other areas of software engineering or other types of learners. We summarize these challenges into three aspects (need to impart practical skills, relevance to the learner's environment and easy to comprehend) and describe how we design our course to address these challenges. We also highlight and compare our teaching methods based on PBL and CBL. Our evaluation on the effectiveness of our course design and the teaching methods for the workshops are based on the adult learners' feedback ratings over a period of 7 years and 28 runs of the course. Our results indicate an encouraging trend of improving ratings for our course as a whole with the adapted PBL method being preferred by our adult learners.

In this paper, we highlight how abstract concepts in software architecture design can be taught in public courses for adult learners and how we have designed our course to meet the practicality and relevance needs for this group of adult learners. Over the years, this course has evolved with the feedbacks of the learners and their organizations and we are glad that it has become one of the essential course to take in these IT organizations. We hope this discussion and insights can support education researchers in investigating better pedagogical approaches for adult learners, help educators design better software architecture curriculum and are useful to educators who encountered similar challenges in teaching abstract concepts.

In our future work, we plan to provide our experiences and insights in designing software architecture courses to another two type of learners – undergraduates and experienced adult learners with more than 10 years of working experiences. We also plan to perform a comparison study to collate our findings in teaching software architecture course to all these three groups of learners and derive further insights.

REFERENCES

[1] P. Kruchten, "The Software Architect," Springer, Boston, MA, 1999, pp. 565-583.

[2] "ANSI/IEEE 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems," 2001. [Online]. Available: https://standards.ieee.org/findstds/standard/1471-2000.html.

[3] T. Mannisto, J. Savolainen and V. Myllarniemi, "Teaching software architecture design.," in *Seventh Working IEEE/IFIP Conference on Software Architecture, WICSA 2008.* , 2008.

[4] E.L. Ouh and S. Jarzabek, "An Adaptability-Driven Model and Tool for Analysis of Service Profitability." In: Nurcan S., Soffer P., Bajec M., Eder J. (eds) Advanced Information Systems Engineering. CAiSE 2016. Lecture Notes in Computer Science, vol 9694. Springer, Cham

[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*, 2015.

[6] 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.

[7] 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."* Academic Medicine, 82(1), 74-82, 2007.

[8] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1998.

[9] P. Kruchten, The rational unified process: an introduction, Addison-Wesley Professional, 2004.

[10] "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.

[11] K. L. Minh, E. L. Ouh, and S. Jarzabek. "Mood self-assessment on smartphones." In Proceedings of the conference on Wireless Health, p. 19. ACM, 2015.

[12] D. Koznov, D. Luciv, H. A. Basit, E. L. Ouh, and M. Smirnov. "Clone detection in reuse of software technical documentation." In International Andrei Ershov Memorial Conference on Perspectives of System Informatics, pp. 170-185. Springer, Cham, 2015.

[13] D. Abrahams and A. Gurtovoy, C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond, Portable Documents, Pearson Education, 2004.

[14] "TOGAF®, an Open Group standard," [Online]. Available: http://www.opengroup.org/subjectareas/enterprise/togaf.

[15] "Methods, Open Group - List of Recognized Methods," [Online]. Available: http://www.opengroup.org/openca/cert/methods.tpl

[16] Analysis ToolPak Add-in for Microsoft Excel [Online]. Available: https://support.office.com/en-us/article/use-the-analysis-toolpak-to-perform-complex-data-analysis-6c67ccf0-f4a9-487c-8dec-bdb5a2cefab6

[17] Amazon Comprehend - Discover insights and relationships in text [Online]. Available: https://aws.amazon.com/comprehend/