Teaching SOLID Software Design Principles Using Peer Instruction—A Pilot Study

Dr. Bhuvaneswari Gopal, University of Nebraska, Lincoln

Dr. Bhuvaneswari (Bhuvana) Gopal is an Assistant Professor of Practice in the School of Computing at the University of Nebraska-Lincoln where she teaches Software Engineering, Software Security in Practice, the industry internship course, and leads the Learning Assistant Program that serves several computing courses at the School of Computing. Dr. Gopal has extensive experience in the software industry, where she spent 14 years in various roles, including Software Architect and Lead software engineer positions before switching to full time academia. She is also a Microsoft certified professional, with an MCPD certification. Her research work is focused on evidence-based, active learning pedagogies to improve software engineering education. She has published papers specifically on Peer Instruction and Process Oriented Guided Inquiry-based Learning-like pedagogical approaches in the undergraduate software engineering classroom, both in person and virtual. She holds a Bachelors degree in Physics from the University of Madras, Chennai, India. She holds two Masters degrees - one in Computer Science from the University of Nebraska-Lincoln, and one in Medical Physics from Anna University, Chennai, India. She holds a Ph.D. in Computer Science from the University of Nebraska-Lincoln.

Teaching SOLID software design principles using Peer Instruction – A pilot study

Bhuvaneswari Gopal
School of Computing, University of Nebraska-Lincoln

Abstract

In this paper we study the efficacy of Peer Instruction (PI) as a pedagogy for teaching the SOLID principles of software design in an in-person undergraduate software engineering classroom. SOLID principle based software design is an important topic in helping students get ready to work in the software industry. Peer Instruction (PI) is an active learning pedagogy in which students actively participate in their own learning by discussing questions with small groups of peers during class, providing real-time feedback to the instructor through an online portal, or a handheld student response system device, or through manual raising of hands or flashcards. Several studies on PI in computing exist, but very few studies focus on PI in the topic of software design in general and/or SOLID principles in particular. In this pilot study we focus on comparing correctness gains for students learning SOLID principles through lecture-based instruction and PI. We utilized a purely lecture based approach in one semester, followed by PI in the next semester. We developed our own PI questions for SOLID principles, which is another unique contribution of this study. We conducted pre- and post-course surveys for both the lecture iteration as well as the PI iteration of the course. We analyzed over 120 student responses to the pre- and post- surveys to determine if PI had helped students learn and recognize SOLID principles more effectively than lectures. Cognitively, we found a correlation between PI and student learning, by observing encouraging increases in levels of success as measured through cognitive pre- and post-course survey, for SOLID principles. We found statistically significant correctness gains for students with PI over lectures, indicating that students learned better using PI than they did through lectures. We categorized our findings regarding the student population in general, as well as in specific underrepresented minorities in computing. In this regard, we found that PI helped students of marginalized identities, specifically students of color, women, and first-generation students in computing statistically significantly more than lectures did, with their cognitive gains. From a SOLID principles perspective, PI also seemed to have helped students with little to no prior internship experiences do at least as well as students with one or more industry internships leading up to the course.

Keywords

faculty paper, Peer Instruction, software architecture and design, active learning, software engineering

Introduction

Peer Instruction (PI), an active learning approach, has gained attention from the computing education community over the last few years [1]. The focus in PI is active student engagement through discussion, involving students in the answering and discussion of multiple-choice

questions. This is typically accomplished by obtaining real-time student feedback through the use of student response systems in class as the students learn the topic.

SOLID is an acronym that denotes five basic principles widely used in designing software built on the .NET platform. S stands for SRP (Single Responsibility Principle), O for OCP (Open Closed Principle) L for LSP (Liskov Substitution Principle), I for ISP (Interface Segregation Principle) D for DI (Dependency Inversion Principle). The main purpose of these principles is to help software design withstand volatility and change, making these the cornerstone of Agile .NET software design.

In this paper, we seek to understand if, in teaching SOLID principles, computing related majors such as Computer Science (CS), Computer Engineering (CE) and Software Engineering (SE) would benefit from utilizing PI as the pedagogy compared to pure lecture based instruction. We used a pre- and post-survey study design to measure student learning across three class sessions for the SOLID principles. Each of these class sessions was conducted with PI as the primary mode of instruction.

The rest of the paper is organized as follows. Section II describes the related literature. Section III details the implementation of PI and the research methods that we used in this study. Section IV and Section V present and analyze our results, Section VI presents possible threats to the validity of our study, and Section VII delineates possible future work and concludes this paper.

Related Literature

This paper focuses on two distinct topics: SOLID principles, which is a topic pertaining to software architecture, and PI, which is an active learning, small group-based pedagogical approach. As such, prior literature pertaining to both topics in the context of teaching CS and SE is explored in this section.

Teaching software architecture

The need for robust software architecture and design is outlined in the set of guidelines introduced by Robert Martin [2] called Design Principles. Software can be fragile, viscous, immobile and rigid if not designed and implemented properly. These are factors that software engineers need to take into account while they consider the architecture of the proposed software system. These factors also make teaching software architecture a difficult endeavor.

There are several documented difficulties in teaching software architecture. Experience is the best way to understand the various nuances involved in real world implementations of software architecture, due to its abstract nature. Software architecture is also complex: the aspects of people, process and technology that permeate software engineering are important considerations in software architecture as well. Students need to gain proficiency in the social as well as design / technological implementation aspects of software architecture. To overcome these teaching challenges, Van Deursen et al. [3] developed the “Collaborative Software Architecture Course”. The focus in their course was students working with large open-source systems both with other students as well as architects from such systems. Utilizing open and free communication with

transparency, the course resulted in an online book that was published. The findings from their study were that open-source systems can be successfully used to let students gain experience with key software architecture concepts, and that both students and integrators (architects) from open-source systems were willing to work together seamlessly to benefit both parties.

Vygotsky's [4] zone of proximal development (ZPD) describes the region between the student's individual problem-solving ability and the level that individual can reach in collaboration with more capable peers. Collaborative approaches utilize social constructivist paradigms such as the ZPD to create pedagogy that engages students with each other and the instructor. The studies outlined next utilize collaborative learning environments for software architecture. The study by Lieh [5] presented challenges encountered in teaching adult students software architecture, echoing findings by Galster [6] and de Boer [7]. This study utilized Problem Based Learning (PBL) and Case Based Learning (CBL) and found that adult learners preferred PBL over CBL for instruction on software architecture. Linder, Abbott and Fromberger [8] utilized an instructional scaffolding approach to teaching software design using Extreme Programming (XP).

The lowest levels of abstraction in teaching software architecture are usually done in introductory courses. However, at the introductory course level, it is difficult to help students understand how relevant SE is in society, and what role software architecture plays in SE. These issues have led to over a third of the students dropping CS majors [9]. Often, real-world design problems involve well thought through, nuanced solutions, and an effective software designer must have the ability to utilize people, processes and technologies together to create these solutions. Mannisto, Savolainen, and Myllarniemi [9] described an academic software architecture design course that teaches students to solve demanding design problems with cross-cutting architectural concerns in a manner that is industrially relevant. Their industry-ready course was well received by students.

Lago and van Vliet refer to software architecture as being a wicked problem [10] mainly because there is no one-size-fits-all solution, and tradeoffs need to be made. Galster and Angelov [6] detail these and other challenges in teaching software architecture. De Boer et al. [7] explain that the nature of software architecture, and the "wickedness" of the problem without a single unique solution approach, cannot be met with in traditional lectures where students are passive. They argue that a collaborative learning approach is imperative. The work by Paulisch [11] agree with the pedagogical approaches described by Jeff Offutt [12] which advocates for allowing students to develop "divergent thinking," again circling back to the problem of software architecture not being able to provide a single, unique, one size fits all, "correct" solution to any given problem.

PI in computer science and software engineering

PI is fast becoming a popular active learning, student centric pedagogy in CS education. PI was developed by Mazur [1] and originally adopted in Physics classrooms. PI has been utilized heavily in the hard science disciplines over the last three decades and is now gaining widespread acceptance in engineering disciplines including CS and Computer Engineering (CE). Several pilot and replication studies have established the effectiveness of PI as the pedagogy of choice for teaching CS courses. Utilizing PI, several studies have been conducted in CS; chief among

them being the pioneering studies by Porter, Cutts and Simon [12-17]. How students learn from discussions resulting from in-class multiple choice questions and the extent to which this learning manifests on a final exam have been studied in several introductory CS courses [12-14], thus correlating the increase in in-class correctness due to PI to the single final exam to find statistically significant relationships. PI has been utilized in teaching introductory level CS classes as well as cybersecurity [18,19].

In software engineering (SE), the body of work by Esper [20], Adawi [21], and Gopal and Cooper [22-25] has laid the foundation for implementing PI in undergraduate SE classrooms. Gopal and Cooper also specifically examined possible relationships between in-class student answer correctness, and quiz and exam performance in an undergraduate software engineering course [24]. Herman and Azad [26] compared PI and collaborative learning in an undergraduate computer architecture course. Their findings suggested that PI improved student learning efficiency, saving students time, and reducing student stress, in addition to increasing their sense of belonging. Utilizing PI to teach software architecture in general, and SOLID principles in particular is a nascent sub field in CS education. Our work in this study combines teaching SOLID principles of software design with the active learning pedagogy of PI. In addition to implementing PI in the classroom for the industry relevant topic of SOLID principles in software architecture, we also developed PI materials that could be used by other instructors who wish to do the same [27].

Research Questions

In this paper, we describe a study using PI in an undergraduate SE class specifically on the topic of SOLID principles, utilizing the online student response system on iClicker [28]. Our research questions for this study were:

RQ1: In an undergraduate SE class, does PI help students learn the concepts of SOLID principles better than students in a purely lecture format class?
RQ2: In an undergraduate SE class, does PI help students belonging to underrepresented groups in CS learn the concepts of SOLID principles better than their purely lecture format counterparts?
RQ3: In an undergraduate SE class, does PI help students with no prior internships of any kind in CS learn the concepts of SOLID principles as well as students who have had at least one internship prior to the class?

Classroom implementation of PI and PI question development

Our classroom implementation of PI followed the steps outlined by Porter et al. [13], and Gopal and Cooper [22]. The sequence of steps we followed consisted of preparatory readings as part of the flipped classroom approach. These readings were assigned to students to be completed before the class session. During class, mini lecture sessions were interspersed with students actively engaged with multiple choice questions designed to help them confront and explore challenging concepts [22]. Students used the online student response system through iClicker [28] to record their real time responses. Students answered twice, with an initial vote before peer discussion, and a final vote after a 2-3-minute discussion with small peer discussion groups of 3 students

each. After the final vote the instructor, who is the author of this paper, held a short class wide discussion, at the end of which the correct answer to the clicker question was clearly indicated to the entire class.

PI utilizes isomorphic questions [14]. We developed a set of isomorphic PI questions for SOLID principles. Our PI questions are available at the website [27]. The development of PI questions involved multiple steps.
Step 1: We identified where students had common misconceptions in the topic.
Step 2: We determined the number of questions we wished to include in each class session, based on the typical length of a PI cycle in class [1].
Step 3: In accordance with the flipped classroom approach [1] for PI, our third step was to decide what material needed to be assigned to students for familiarization prior to class, and what material would be based on the mini-lectures.
Step 4: We collected meaningful distractors for each question, and formulated multiple choice questions for the letters of sub-topic within the topic of SOLID principles. Specifically for SOLID principles, we developed two questions each for the Single Responsibility Principle (S), Open Closed Principle (O), Liskov Substitution principle (L), Interface Segregation Principle (I) and Dependency Inversion Principle (D).

We ensured that each question had one single correct answer, but that the questions encouraged students to think beyond obvious answer choices. We also ensured that the PI questions spurred students to discuss possible answer choices by making distractors realistic. Our goal was to formulate questions that required students to engage at the higher levels of Bloom's taxonomy, where they synthesize and analyze allied concepts to answer the given question, and not simply recall from memory [29].

Survey Design and Methodology

This research project was reviewed and determined to be exempt by our college's Institutional Review Board (IRB). Our experimental setup consisted of two groups of students at a large Midwestern R1 University, in an undergraduate, pre-capstone SE course. We utilized a quasi-experimental pretest-posttest hybrid between groups and within groups design for this study. The control and treatment groups consisted of successive cohorts of sophomores/juniors from CS and Computer Engineering, one section each. This SE course was a mandatory component of their academic progression towards earning their degree.

The treatment group was taught using PI while the control group received instruction through traditional lectures. The treatment group and control group were instructed in consecutive semesters by the same instructor with the same content. We created a survey questionnaire consisting of questions based on SOLID principles. We administered the survey questionnaire to students at the beginning and again at the end of the course. The survey questionnaire consisted of ten "True/False/Neither True nor False" questions, based on the SOLID principles content taught in class. The end of the survey contained four questions on student demographics including gender identity, college standing, first generation status, and information on prior software internships. Both the control and treatment groups were administered this questionnaire at the start of and at the end of the semester, and student responses were collected. Students were

not compensated for their participation in the surveys. There were 63 students in the treatment group and 58 students in the control group.

The cognitive survey questionnaire contained the following questions, each with three answer choices – true, False, Neither True nor False.
Q1 - Every object should have a single responsibility, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility.
Q2 - This is an example of the Single Responsibility Principle (SRP): class that compiles and prints a report.
Q3 - When adding behavior, SRP helps us clearly decide whether to extend the object or create another object.
Q4 - SRP may make code more complex and decomposed for an eventuality that never comes.
Q5 - Open/Closed principle is followed through the use of Interfaces, where multiple implementations could be created and polymorphically substituted for one another.
Q6 - For the Open/Closed principle, existing interface is closed to modification and new implementations must, at a minimum, implement that interface.
Q7 - For the Liskov substitution principle, subtypes may not introduce methods that are not present in the supertype.
Q8 - Large, complex interfaces help implement Interface segregation principle.
Q9 - For Interface segregation principle, if you need to change one interface, you should change the other.
Q10 - For Dependency Inversion principle, high-level modules should depend on low-level modules.

Results

We found that students in the treatment group (the group that was instructed using PI) had statistically significant increases in scores on the post- test compared to the pre- test, while the students in the control group (the group that was instructed using lectures) did not.

There were two “between groups” distributions (pre-control vs pre-treatment; post-control vs post-treatment) and two “within groups” distributions (pre-control vs post-control; pre-treatment vs post-treatment). We analyzed the scores data from each question for normality and found the skew of each of the distributions was close to zero. For each distribution, the kurtosis was also inside the acceptable range for a normally distributed curve, as was the shape of the curve itself, indicating that all distributions were normally distributed. We ran ANOVA tests on all the distributions to determine statistical significance since the assumption of homoscedasticity and normality were met on all the score distributions. We adopted a significance level of $\alpha = 0.05$ to report our results.

Pre-control vs pre-treatment

We compared the scores on the pre survey for the control group versus the treatment group and found that the two groups had starting points that were not dissimilar (pre-test: $\chi 2 = 27.42$, $p = 0.42$). The knowledge levels where the two groups began their journey in learning SOLID principles were not statistically different.

Pre-treatment vs post-treatment

Based on our analysis, we found that students in the treatment group obtained scores that were statistically significantly higher on their post survey compared to the pre survey ($\chi 2 = 93.89$, $p < 0.0001$). Figure 1 shows the distribution of scores in the pre and post-test for the treatment group.

Post-control vs post-treatment

Based on our analysis, we found that students in the treatment group obtained scores that were statistically significantly higher on their post survey compared to the post survey scores of students in the control group ($\chi 2 = 90.01$, $p < 0.0001$). Figure 2 shows the distribution of scores in the post-test for the control and treatment group.

Pre-control vs post-control

Based on our analysis, we found that students in the control group obtained scores that were not statistically significantly higher on their post survey compared to their pre survey scores ($\chi 2 = 36.57$, $p = 0.21$).

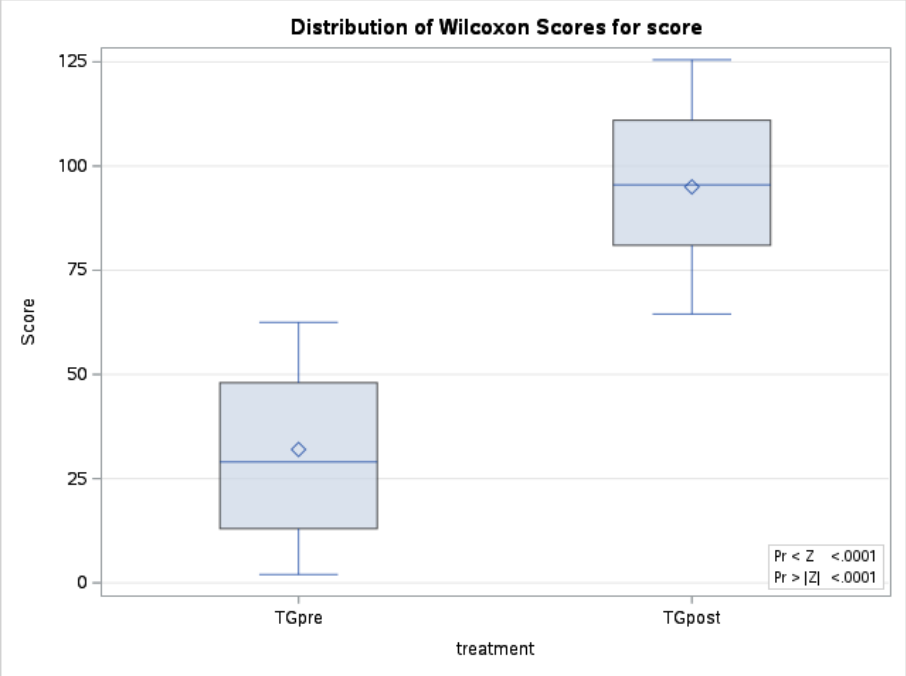

Figure 1. Distribution of scores in the pre and post-test for the treatment group

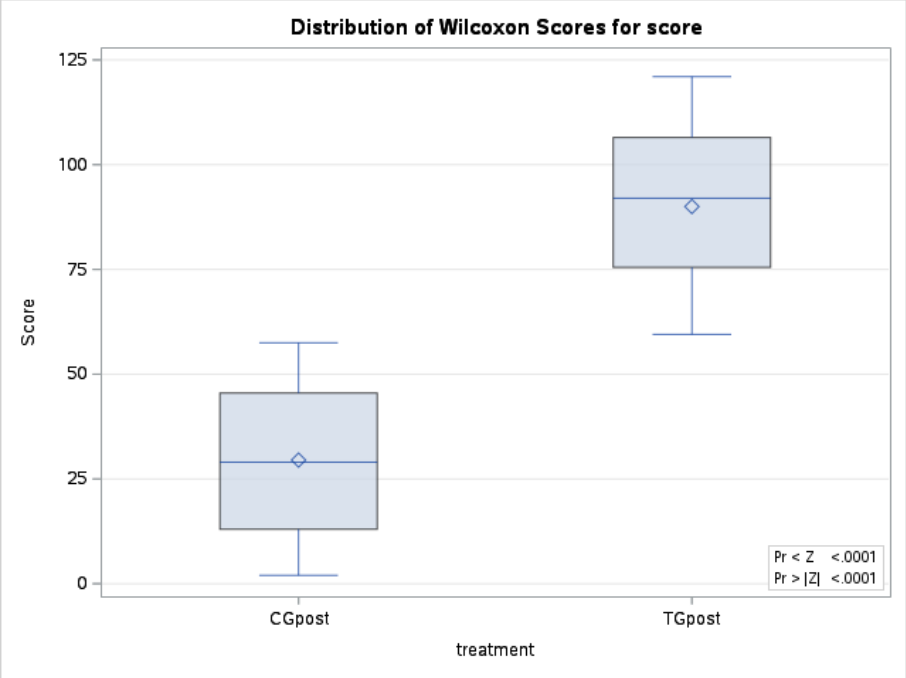

Figure 2. Distribution of scores in the post-test for the control and treatment groups

The demographic and professional experience characteristics of our student sample for the treatment group were determined based on students self identifying themselves under various categories: students of color, women, first generation status (whether or not the student was the first in their family to attend college), and whether or not students had a prior computer science internship.

We categorized our findings regarding the student population in general, as well as in specific underrepresented minorities in computing. In this regard, we found that PI helped students of marginalized identities, specifically students of color, women, and first-generation students in computing statistically significantly more than lectures did, with their cognitive gains. From a SOLID principles perspective, PI also seemed to have helped students with little to no prior internship experiences do at least as well as students with one or more industry internships leading up to the course.

The score distributions for students of color, women, first gen and students without prior internships were not normally distributed, based on running the Shapiro-Wilk test. To analyze these distributions, we utilized non-parametric analysis of variance techniques, specifically the Kruskal-Wallis test.

Students of color, post-control vs post-treatment

Based on our analysis, we found that students of color in the treatment group obtained scores that were statistically significantly higher on their post survey compared to their control group peers ($\chi 2 = 35.21$, $p < 0.01$).

Women in computing, post-control vs post-treatment

Based on our analysis, we found that women students in the treatment group obtained scores that were statistically significantly higher on their post survey compared to their control group peers ($\chi 2 = 29.88$, $p < 0.01$).

Effect of internships within the treatment group, post-control vs post-treatment

Based on our analysis, we found that students in the treatment group who did not have any prior computing internships did at least as well as the students who had prior internships, based on their post survey scores ($\chi 2 = 0.24$, $p = 0.78$). With the pure lecture class, we observed a statistically significant difference ($\chi 2 = 34.22$, $p < 0.001$).

Discussion

The overall takeaway from our findings is that students in the PI group learned the concepts of SOLID principles better than their lecture group counterparts, based on their score gains. These gains permeated through several sections of students – women, students of color, and students without prior internships. PI has been shown in previous studies to improve learning gains in several topics [14-16, 22-24], but specific analysis on women and students of color has been lacking, and our study attempts to bridge the gap.

Prior experience in CS is an important factor in fostering belonging and predicting persistence in the discipline [30]. Specifically in software engineering which is a heavily industry based sub-discipline within CS, industry internships offer students valuable hands-on experience in applying CS concepts in the real world. Our findings indicate that PI leveled the playing field when it came to students with internships and students without any. Students in the treatment group who did not have any prior internships did at least as well as the students with one or more prior industry internships. When both sets of students underwent instruction through traditional lecture, students with internships fared better in the post-test compared to those students who had no prior internship experience. This indicates that in our study, PI was clearly a significant factor for students with no prior experience to have done at least as well as the students with prior internships. One reason for this could be the intrinsic nature of PI groups. Students with prior internships tend to have more confidence in approaching the subject [30]. When forming PI groups, there could be a combination of such students along with students with no prior experience. This could have afforded students the opportunity to share their insights from prior internships freely during PI discussions. In a topic like software architecture and design, different industry companies implement concepts in their own unique way. PI discussions could have helped students with internships recall their experiences or observations from their internships, and share them in context with their peers who had no prior internship experience.

Several studies exist in literature explaining the challenges and difficulties faced by women and students of color in CS [31-34]. Confidence and sense of belonging are challenges encountered by underrepresented minorities in computing, resulting in a negative impact on retention. These studies also offer several findings on what improves success for these student groups who are traditionally more likely to struggle in CS. Chiefly, the takeaways from these studies indicate

that representation matters, role models matter, and teaching methods need to be geared towards collective success, not just individual achievement [32]. PI is a socially constructivist pedagogical approach, allowing students to co-construct knowledge instead of being told what to learn [1, 14-16]. When students are able to discuss questions with other students in short, focused bursts of time, where there is no pressure to earn a grade, they can speak their mind freely while discussing the topic, thus potentially increasing confidence levels in the topic. Prior research has shown the positive impact that PI has on student affect, specifically, gender, interest, confidence, and professional perception [22,25]. We think that these characteristics of PI have led us to our findings, which show that women and students of color performed better when being instructed using PI when compared to traditional lectures.

Threats to validity

As with any empirical pilot study, there are a few threats to the validity of our findings. Our student sample size is relatively small (a total of 121 students across both semesters), since we only have one pair of classes- one each for the control and treatment groups. Having the two groups in successive semesters (Spring vs Fall) could be a confounding variable, particularly with teaching strategies potentially varying in these semesters. Another potential confounding variable is class timing (morning/afternoon/evening). However, we tried to minimize the effect of these threats as follows. Both semesters were taught by the same instructor, with the same course materials in the same order of topics, during the same class timing each semester. The only difference between the control and treatment groups was the method of teaching, with the control group being taught strictly through a traditional lecture with slides, while the treatment group was taught with the classroom implementation of PI outlined in this paper.

Conclusion and Future Work

We have presented, in this paper, the findings from our pilot study on utilizing PI to instruct sophomore/junior level undergraduate computer science and engineering students in the topic of SOLID principles in software engineering. We utilized a quasi-experimental setup for our intervention. Students in the PI class obtained statistically significant score gains compared to the students in the purely lecture format class. PI also helped students of color, women, and first-generation students score statistically significantly higher than their lecture class counterparts. With PI, students who never had any internships scored at least as well as students with one or more internships, compared to the students in the pure lecture class. In this context, we conclude that student learning with SOLID principles is impacted positively across several student categories.

RQ1: In an undergraduate SE class, we found that PI does help students learn the concepts of SOLID principles better than students in a purely lecture format class.
RQ2: In an undergraduate SE class, we found that PI does help students belonging to underrepresented groups in CS learn the concepts of SOLID principles better than their purely lecture format counterparts.
RQ3: In an undergraduate SE class, PI does help students with no prior internships of any kind in CS learn the concepts of SOLID principles as well as students who have had at least one internship prior to the class.

The encouraging results on student success from this study show that PI has helped students belonging to various categories learn the concepts behind SOLID principles. This success emphasizes the need for further research on the specific aspects of PI as a pedagogical approach that contribute towards student success. In future work, we wish to further validate our findings from this pilot study by replicating and extending this work. We also hope to delve deeper into PI answer patterns and connect them with student discussions. We intend to conduct longitudinal studies to further our understanding on how PI discussions could impact cognitive gains in teaching software architecture, specifically, SOLID principles. We also hope to investigate how PI compares with other student centric, small group pedagogies, in teaching software architecture principles.

References

[1] E. Mazur, "Peer instruction: Getting students to think in class.," in AIP conference proceedings (Vol. 399, No. 1). American Institute of Physics., Mar. 1997, pp. 981-988.

[2] R. C. Martin, "Design principles and design patterns.," in Object Mentor, 1(34), 2000, p.597.

[3] A. Van Deursen, M. Aniche, J. Aué, R. Slag, M. De Jong, A. Nederlof, and E. Bouwers, "A collaborative approach to teaching software architecture", in Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education", Mar. 2017, pp. 591-596.

[4] L. S. Vygotsky, and M. Cole, "Mind in society: Development of higher psychological processes.", Harvard university press, 1978.

[5] O.E. Lieh, and Y. Irawan, "Teaching adult learners on software architecture design skills.", in 2018 IEEE Frontiers in Education Conference (FIE), Oct. 2018, pp. 1-9.

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

[7] R.C. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen, "Architectural knowledge: Getting to the core.", in Software Architectures, Components, and Applications: Third International Conference on Quality of Software Architectures, QoSA 2007, Medford, MA, USA, Revised Selected Papers 3, Springer Berlin Heidelberg, Jul. 2007, pp. 197-214.

[8] S. P. Linder, D. Abbott, and M.J. Fromberger, "An instructional scaffolding approach to teaching software design", in. Journal of Computing Sciences in Colleges, 21(6), 2006, pp.238-250.

[9] T. Mannisto, J. Savolainen and V. Myllarniemi, “Teaching software architecture design.”, in Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Feb. 2008, pp. 117-124.

[10] P. Lago, and H. Van Vliet, “Explicit assumptions enrich architectural models.”, in Proceedings of the 27th international Conference on Software Engineering, May. 2005, pp. 206-214.

[11] F. Paulisch, M. Backert and T. Blum, "Lessons Learned From a Learning Program for Software Architects," in *IEEE Software*, vol. 40, no. 6, Nov.-Dec. 2023, pp. 55-62.

[12] J. Offutt, “Putting the engineering into software engineering education.”, in IEEE software, 30(1), 2013, pp.96-96.

[13] C.B. Lee, S. Garcia, and L. Porter, “Can peer instruction be effective in upper-division computer science courses?”, in ACM Transactions on Computing Education (TOCE), 13(3), 2013, pp. 1-22.

[14] L. Porter, D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro and B. Simon, “ A multi-institutional study of peer instruction in introductory computing.”, in Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Feb. 2016, pp. 358-363.

[15] L. Porter, C.B. Lee, B. Simon, Q. Cutts, and D. Zingaro, D., “Experience report: a multi-classroom report on the value of peer instruction.”, in Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, Jun. 2011, pp. 138-142.

[16] L. Porter, C.B. Lee, B. Simon, and D. Zingaro, D., “Peer instruction: Do students really learn from peer discussion in computing?.”, in Proceedings of the seventh international workshop on Computing education research, Aug. 2011, pp. 45-52.

[17] B. Simon, S. Esper, L. Porter, and Q. Cutts, “Student experience in a student-centered peer instruction classroom.”, in Proceedings of the ninth annual international ACM conference on International computing education research, Aug. 2013, pp. 129-136.

[18] P. Deshpande, C.B. Lee, and I. Ahmed, “Evaluation of peer instruction for cybersecurity education.”, in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Feb. 2019, pp. 720-725.

[19] W.E. Johnson, A. Luzader, I. Ahmed, V. Roussev, G.G. Richard III, and C.B. Lee, “Development of peer instruction questions for cybersecurity education.”, in 2016 USENIX Workshop on Advances in Security Education (ASE 16), 2016.

[20] S. Esper, “A discussion on adopting peer instruction in a course focused on risk management.”, in Journal of Computing Sciences in Colleges, 29(4), 2014, pp.175-182.

[21] T. Adawi, H. Burden, D. Olsson, and R. Mattiasson, “Characterizing software engineering students’ discussions during peer instruction: Opportunities for learning and implications for teaching.”, in International Journal of Engineering Education, 32(2), 2016, pp. 927-936.

[22] B. Gopal, S. Cooper, “Peer instruction in software testing and continuous integration.”, in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Mar. 2021, pp. 548-554.

[23] B. Gopal, S. Cooper, “Peer Instruction in Online Synchronous Software Engineering-Findings from fine-grained clicker data”, in 2021 IEEE Frontiers in Education Conference (FIE), Oct. 2021, pp. 1-8.

[24] B. Gopal, S. Cooper, “Peer instruction in software engineering-findings from fine-grained clicker data”, in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Mar. 2021, pp. 115-121.

[25] B. Gopal, S. Cooper, “Peer instruction in online software testing and continuous integration: A replication study.”, in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training, May 2022, pp. 199-204.

[26] G.L. Herman, and S. Azad, “A comparison of peer instruction and collaborative problem solving in a computer architecture course.”, in Proceedings of the 51st ACM Technical Symposium on Computer Science Education, Feb. 2022, pp. 461-467.

[27] B.Gopal, “Peer Instruction questions for SOLID principles”, https://cse.unl.edu/~bgopal/PISOLID.htm, (accessed Dec. 2023).

[28] iClicker. “Student response systems & classroom engagement tools.”, https://www.iclicker.com/ (accessed Dec. 2023).

[29] D.R. Krathwohl, “A revision of Bloom's taxonomy: An overview.”, in Theory into practice, 41(4), Nov. 2002, pp 212-218.

[30] M. Minnes, S.G. Serslev, and O. Padilla, O., “What do CS students value in industry internships?.”, ACM Transactions on Computing Education (TOCE), 21(1), 2021, pp.1-15.

[31] S.R. Johnson, A. Ivey, J. Snyder, M. Skorodinsky, and J. Goode, “Intersectional perspectives on teaching: Women of color, equity, and computer science.”, in 2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), Mar. 2020, (Vol. 1, pp. 1-4).

[32] A. Johnson, G.C. Townsend, and K. Stewart, “Students of Color Organization Improves CS1 Grades.”, in Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2, Mar. 2022, pp. 1085-1085.

[33] Palmer, R.T., Maramba, D.C. and T.E. Dancy II, “A qualitative investigation of factors promoting the retention and persistence of students of color in STEM.”, Journal of Negro Education, 80(4), 2011, pp. 491-504.

[34] G.C. Townsend, K. Stewart, and A. Johnson, “Recruiting students of color through a student organization.”, in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Mar. 2021, pp. 1249-1249.