Reflections on Teaching Refactoring: A Tale of Two Projects

Shamsa Abid
Computer Science Department
SBA School of Science and Engg.
Lahore Uni. Of Mgmt. Sciences
(0092) 42 3560 8194
shamsaabid123@gmail.com

Hamid Abdul Basit
Computer Science Department
SBA School of Science and Engg.
Lahore Uni. Of Mgmt. Sciences
(0092) 42 3560 8194
hamidb@lums.edu.pk

Naveed Arshad
Computer Science Department
SBA School of Science and Engg.
Lahore Uni. Of Mgmt. Sciences
(0092) 42 3560 8194
naveedarshad@lums.edu.pk

ABSTRACT
Teaching refactoring effectively while making students realize the importance and benefits of refactoring is a challenge. In this direction, an experiment was carried out while conducting the course project for the Refactoring and Design Patterns course. This paper discusses the results of the experiment that involved two different project schemes to carry out refactoring activities on the same code base. One scheme was *post-enhancement refactoring* and the other was *pre-enhancement refactoring*. The aim of the experiment was to decide which scheme was beneficial in terms of better understanding, appreciation, and implementation of refactoring.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement - *restructuring, reverse engineering, and reengineering.* K.3.2 [**Computer Science Education**] Computer and Information Science Education – *computer science education, curriculum.* K.6.1 [**Management of Computing and Information Systems**]: Project and People Management - *life cycle*, Software Management – *software development, software maintenance.*

General Terms
Management, Measurement, Design, Experimentation, Human Factors

Keywords
Software refactoring, teaching refactoring, refactoring course project, student survey

1. INTRODUCTION
Refactoring is the process of transforming the internal structure of existing code, while keeping its functionality intact. It improves the design of code and makes it easier to maintain and understand [4]. Software refactoring is a highly desirable activity for good quality maintainable software [5], and for increased developers' productivity [6]. In the past few years, software development has shifted from traditional software processes to agile software development. According to recent industry surveys[1][2], agile methodologies are being used in almost three quarters of software development projects now. Since agile development stresses on working software and shorter release cycles with less upfront design, refactoring is a cornerstone of these methodologies. To this end, it is important to cultivate the realization of the importance and benefits of refactoring in aspiring but novice developers. This responsibility of nurturing developers with the right principles and practices falls on the academia; wherein lies the challenge of figuring out the best approach to teach refactoring, which enhances students' understanding and appreciation of refactoring.

In this regard, an experiment was conducted to compare two schemes of carrying out the course project for the Refactoring and Design Patterns course, offered to Computer Science graduate and senior undergraduate students at Lahore University of Management Sciences. The Refactoring and Design Patterns course includes the teaching of software design patterns, design principles, and refactoring practices, which are then applied on the course project by the students. The course project is a crucial and practical part of the course. It comprises of refactoring a given code base, with the intent of improving the structure and design of the code with respect to its readability, maintainability, and changeability.

The experiment was conducted on two groups of students (23 in all) who were required to work on one of the two different project schemes. In the first scheme, which we call *post-enhancement refactoring*, the students were asked to first perform a series of functional enhancements on the provided code. After that, they were required to perform an analysis of code quality metrics and code smells on the resultant code, and follow it up by refactoring the identified code smells. *Code smells* are a metaphor to describe code patterns that are generally associated with bad design and bad programming practices, and indicate that refactoring can be applied [4][9]. In the second project scheme, which we call *pre-enhancement refactoring*, the students were asked to first analyze the code quality metrics and code smells, and refactor those smells. After that, they were asked to perform a series of functional enhancements on the refactored code. Both project schemes had a final phase of application of design patterns, after the enhancement and refactoring phases had been completed. The two project schemes are illustrated in Figure 1. The two schemes vary only in terms of ordering of project phases; whether refactoring is done prior to, or after the enhancement phase.

ITICSE'15, July 4–8, 2015, Vilnius, Lithuania.

DOI: http://dx.doi.org/10.1145/2729094.2742617

[1] https://www.planbox.com/blog/agile/scrum/research/2013-Study-reveals-Statistics-on-Agile-Market-Share.html
[2] http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

The objective of the experiment was to evaluate the performance and feedback of the students working on the two different project schemes. By evaluating the performance and feedback of the students, we were able to make conclusions as to which project scheme resulted in a better performance of students; in terms of better quality refactorings, better quality of resulting code, better understandability, and better appreciation of refactoring.

In this paper, we address the following research questions:

RQ1: What is the better scheme to teach refactoring? Should students perform refactoring first or enhancement first?

We compare the teaching effectiveness of the two project schemes with respect to understanding and appreciation of refactoring. We draw conclusions after detailed analysis of students' performance and evaluation of their project deliverables.

RQ2: How to make students realize the benefit of refactoring?

We investigate which project scheme was better able to highlight the benefits of refactoring. The answer is found through the feedback obtained from the surveys conducted.

RQ3: Which student group is more satisfied?

To answer this, we analyze which project scheme is personally favored by students, by analyzing students' feedback.

The remainder of this paper is organized as follows. Section 2 describes the related work. In Section 3, we discuss the experimental setup for our research methodology. We give details on how the course project was carried out, and how the data was collected. In Section 4, we present the results of our analysis and discuss the threats to validity. Section 5 concludes the paper and mentions future work.

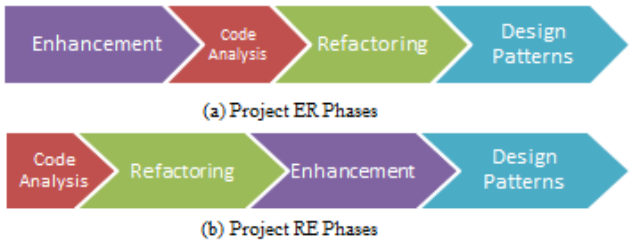

Figure 1. A comparison of two project schemes

2. RELATED WORK

To the best of our knowledge, our study is the first of its kind to compare two approaches to conducting a refactoring course project, with the intent of enabling better understanding, appreciation, and implementation of refactoring techniques among students.

Work has been done in the domain of teaching refactoring with innovative techniques. One such technique by Smith et. al., [1] recommends structuring the lesson plans in a way that every lesson plan would cover a few refactorings, which would be performed as a hands-on activity incrementally as the lessons proceed. In our course, the students are expected to perform refactorings studied in class on their project as the lectures proceed. They are not restricted to a certain set of refactorings, and depending on the code smells, they are free to perform relevant refactorings. The students working on enhancements are free to perform opportunistic refactoring wherever the need arises. Students are given the flexibility to choose to refactor while working on enhancements. Their technique also contrasts with ours in the use of enhancement activity to make the students realize the need for the refactored code.

Dibble and Getswicki [2] advocates the need for dual analysis using automated and manual methods. We also require students to first perform a full fledge analysis of code smells using automated and manual inspection techniques, and then perform refactoring on the problem areas detected.

Murphy-Hill and Parnin [7] have observed that refactoring is often interleaved with changes to a project's functionality. Their observation was that refactoring was seldom done exclusively; rather it was mostly performed in conjunction with other code changes. In view of this observation, it was expected that our students working on the *pre-enhancement refactoring* scheme would write refactored code while working on the enhancements. However, the deadlines forced them to abandon refactoring entirely during enhancement.

It was observed by DeMeyer et. al. [1] that having a good example to teach refactoring is important, and they used a LAN Simulation example to work on, using a variety of refactoring tools. They propose using that example as a benchmark and common example for students and researchers, suggesting its benefits of being small enough to be easily understood, yet representative of real-world tasks. El_Ramly [3] has shared similar experiences in teaching a software re-engineering course, and emphasized the need for real industrial examples for better understanding and appreciation of re-engineering concepts among students. From the students' feedback, we have also felt that students need to work on a real industrial project to better understand the implications of changing an unknown legacy code. Our approach also gives the students exposure to a real industrial example to work on. The use of a real life industrial example served the purpose of better appreciation of refactoring techniques among students.

3. EXPERIMENTAL SETUP AND EXECUTION

3.1 Course Project Details

3.1.1 Project Domain

To conduct this course project, we used a real-life software from the industry - with appropriate permissions. It is a smart phone application implementing a Goods Collection System. The same code was provided to all the students. It was confirmed that the code had enough refactoring opportunities by the Teaching Assistant (TA) of the course, who was among the developers of the application.

3.1.2 Two Project Schemes

The experiment involved the execution of two different project schemes: Project ER was done with the *enhancement followed by refactoring* scheme (post-enhancement refactoring), whereas Project RE followed the *refactoring followed by enhancement* scheme (pre-enhancement refactoring). One group of student teams worked on Project ER, while the other worked on Project RE. The students formed their own teams and were randomly assigned a project scheme. Each team had two to three members each. Each team worked independently on their projects. The functional enhancements required in Project ER were different from those of Project RE; however, they were of the same difficulty level.

3.1.3 Weekly Meetings

Weekly assessment meetings were held not only to assist the students with their problems but also to monitor their progress actively. It was felt very useful to talk to the students during their projects to know what kind of issues they were facing, and how well their work was progressing.

3.2 Project Phases and Milestones

The phases of each project scheme were clearly defined and distributed across a fixed timeline. During the execution of the projects, some deadlines had to be extended according to students' demand. Following is a breakdown of the work carried out across the major phases, and the project evaluation activities.

3.2.1 Pre-Project Survey

On the very first day of the course, a survey was conducted to assess the technical skills of the students, and their industrial and academic background. It was important to see if all the students were sufficiently equipped with the programming skills to be able to work on the course project. Also, an initial opinion of the students' perception of the importance of refactoring was obtained, and it was found that everyone agreed on its importance at the very start of the course.

3.2.2 Project Setup and Transfer of Domain Knowledge

The TA transferred the domain knowledge through a demo and a tutorial at the start of the project. The running application was shown during the demo, and functional specifications for Project ER enhancement requirements were shared and discussed. Students not familiar with *Eclipse IDE*[3] and *Android SDK*[4] were given a startup tutorial on important topics like setting up these platforms, running the emulator, and debugging etc. The structure of the code was discussed and some important code files were explained in more detail to enable the students to get a better feel of the code. Since almost everyone was familiar with Java, it was assumed that the technology hurdle could be overcome by providing minimal assistance in the enhancement phase.

3.2.3 Project ER (Post-Enhancement Refactoring)

Enhancement (3 weeks)

In this phase, the students doing Project ER were provided with a set of enhancement features to be implemented in the current application. Screenshots of desired screens and use case descriptions were provided for understanding the new requirements. They were supposed to meet the deadline, and were left free to code as they like. It was expected that they would code quick and dirty, and that was what actually happened.

Code smells and statistics (10 days)

In this phase, the students were asked to inspect their code for smells, using both automated and manual detection techniques. They were also asked to compute some of the major code metrics like LOC, complexity, depth, number of classes, number of calls, coupling, statements per method etc. This was meant to serve as a baseline for comparison with their post refactoring code status. Some of the tools used for identifying code smells and code metrics were Sonar Qube[5], SourceMonitor[6], FindBugs[7] and PMD[8]. A UML class diagram of the code was also required, which would enable them to study the associations and level of coupling between the classes.

Refactoring and code statistics (4 weeks)

In this phase, the entire project code had to be properly refactored. The students were asked to refactor the problems identified in the previous phase, using either Eclipse's built-in automated refactoring capability, or some other refactoring tool, or manually. At the end of the phase, they were asked to compare their resulting code metrics with the ones taken earlier to quantify the improvements.

Design Patterns (3 weeks)

In this phase, the students were asked to redesign and reorganize the code, keeping in mind the design principles and design patterns taught in the class. They were asked to identify the problem areas in the existing code which could be improved with the application of suitable design patterns, and then modify the code accordingly.

Final report (4 days)

The students were asked to submit a detailed consolidated project report, documenting work done in all the project phases along with the enhanced, redesigned, refactored, and fully functional code. The reports evaluated the variations in code metrics and code smells after each phase was completed, and conclusions were drawn by observing changes in code metrics across all phases. The students were asked to justify how their new design was more reusable, flexible, and extendable after refactoring.

3.2.4 Project RE (Pre-Enhancement Refactoring)

The phases of Project RE were identical to that of Project ER, only the sequence of refactoring and enhancement phases was interchanged.

Code smells and statistics (1 week)
Refactoring and code statistics (4 weeks)
Enhancement (3 weeks and 5 days)
Design Patterns (3 weeks)
Final Report (4 days)

The students working on Project RE were encouraged to write clean code during their enhancement phase. Furthermore, they were asked to report if they felt their refactoring proved useful to write new enhancements easily. It was expected that some of them would code quick and dirty, and some would perform opportunistic refactoring, and almost all would need to perform a final round of refactoring on the entire code. A final code metrics analysis would reveal if their enhanced code was better or worse than their refactored code.

After culmination all phases of both projects, a comparison was made on the refactorings performed by the two groups.

3.2.5 Pre Refactoring Survey

A survey was conducted a week after the project initiation (while one group was working on their enhancement phase while the other had just completed their code smells detection and metrics analysis phase). This survey questioned students on their opinion on conducting refactoring before or after enhancement, considering a real-life scenario at their workplace. It was asked on

[3] https://eclipse.org
[4] http://www.android.com/
[5] http://www.sonarqube.org/
[6] http://www.campwoodsw.com/sourcemonitor.html
[7] http://findbugs.sourceforge.net/
[8] http://pmd.sourceforge.net/

which factors the decision to refactor first would depend. Their opinion on their value perception towards refactoring for ease of maintenance was also taken. In the end, they were asked to provide the lessons they learnt, and the challenges they faced during their current phase.

3.2.6 Post Refactoring Survey

The third survey was conducted after both groups had finished their refactoring phases. The students were again asked which project scheme they thought was better. Their opinion about the benefits of each scheme was also taken. As a self-assessment of the work performed, they were asked to rate the quality and amount of refactoring they had performed on their project. They were also asked whether the course had motivated them to start refactoring at their workplace, and whether it increased their value perception toward refactoring. In the end, open-ended suggestions and feedback regarding the project were recorded.

3.2.7 Final Presentation

The students delivered a final presentation, detailing the work done throughout the project lifecycle. Together with the course instructor and the TA, an external guest with 5 years of industrial software development experience was also invited as an unbiased evaluator to give feedback regarding the work done by the students. The projects were evaluated on the basis of identification of problems in code, and quality of refactorings performed. It was assessed whether the refactorings contributed significantly to the improvement in code metrics, reduction in code smells, understandability, and ease of maintenance. It was also gauged by the way the students communicated their work, how well the students understood the benefits and application of refactoring techniques.

4. ANALYSIS AND DISCUSSION

To understand and decide which project scheme was better, we analyzed the results of our surveys, performed quantitative analysis of students' code and submitted reports, and used analytical feedback from the industry expert to draw conclusions.

4.1 Pre-Project Survey Results

In the first survey, it was observed that: all of the 23 students were graduate students and had previously taken object oriented programming courses, 16 had medium to high Java proficiency, all of the students had worked on Eclipse, 14 had worked on Android, all of them rated the importance of refactoring as high (even those who were not familiar with the theory and principles of refactoring had a preconceived notion of its importance), 12 claimed to be slightly familiar with the concepts of refactoring and code smells whereas the rest were not familiar, 9 had less than a year industry experience, 6 had 1 to 3 years industry experience, and 7 had 4 to 6 years experience working in the industry. These results ensured that all the experimental subjects had the required level of expertise to enable us to conduct the experiment, without posing a major threat to its validity in terms of outliers.

4.2 Pre Refactoring Survey Results

This survey questioned students on their opinion on conducting refactoring before or after enhancement in a real-life scenario at their workplace. In response, 36% said that they would enhance with refactoring in parallel, and end with full fledge refactoring. The top two factors that affect their decision to refactor turned out to be the project deadline, and whether the code is self-written or not. 91% rated refactoring as highly important. 81% agreed that refactoring improved readability. Surprisingly, the number of students voting for *pre-enhancement refactoring* scheme was *equal* to the number of students voting for *post-enhancement refactoring* scheme. 64% selected that refactoring first gives you a chance to improve code so that smells are not accumulated. 64% selected that enhancement first gives you a chance to understand existing code. 73% agreed that refactoring first would take longer, but it would enable quicker enhancement. 82% thought it was easier to enhance after refactoring, and 55% thought it was easier to refactor after enhancing. In reply to when they would perform refactoring at their workplace, 45% said they would refactor as a personal principle.

Some of the student responses in favor of refactoring first are as follows:

"If you know what you are going to code next, you can keep it in mind while refactoring"

"If everything is clean already, new enhancements are not likely to create mess and it is easier to know where to add which type of code. So, it saves time."

Some of the student responses describing the benefits of doing enhancement first are as follows:

"By wrestling with bad code and then looking at the refactored version we can imagine how easy it (enhancement) could have been."

"There isn't any overhead of doing refactoring again."

4.3 Post Refactoring Survey Results

The third survey was conducted after both groups had finished their refactoring phase. The students were again asked which project scheme they thought was better. Table 1 indicates that Project ER was favored slightly more that Project RE.

Table 1. Students' votes in favor of referred project scheme

	Project ER	Project RE
Total number of students	13	9
Number of responses	10	9
Number of votes in favor	11	8

From Table 1, we were able to derive the answer to *RQ3: Which student group is more satisfied?* By noting that four students from Project RE were in favor of Project ER, whereas three students from Project ER were in favor of Project RE, we can say that Project ER students were more satisfied with their scheme of work.

When asked about the benefits of each scheme, students of Project ER in favor of Project ER scheme expressed their views as follows:

"Doing enhancement first helps in gaining the domain knowledge and code review while we are learning different code smells. We can get to know the code flow, coding conventions, project domain. Once enhancement is completed, we can easily refactor it later."

"It helped me to understand the flow of code first and give me the idea that what are the required refactoring to improve this code."

"My major reason is that in the case of this project, making enhancements in the code gave me a chance to get familiar with the domain and technology of the project which later made it easier for me to perform refactoring."

Following are the comments of Project RE students in favor of Project ER:

"Because when you do enhancement you actually understand code by debugging. After complete understanding of code one can easily refactor. Moreover, in Project RE you have to do refactoring again after enhancement. Project RE approach works if the code is yours, or you understand code to refactor it without any danger of code/functionality loss."

"If we do enhancements first that will make sure better understanding of code and its work flow. During refactoring we mostly tried to follow rules with lesser familiarity with code."

"Since the domain was new to me, writing a code would have given more familiarity to the domain rather than reading it, and/or refactoring it."

Some other findings from the third survey are: 15% respondents were highly satisfied with the quality and amount of refactoring done on their projects while 80% were satisfied on a medium scale, 95% affirmed that the course had motivated them to start refactoring at their workplace, and everyone agreed that it had increased their value perception toward refactoring. The students' unanimous agreement on becoming familiar with the advantages and benefits of refactoring answered *RQ2. How to make students realize the benefit of refactoring?* It was seen that both project schemes were equally effective in making students realize and appreciate the benefits of refactoring.

4.4 Quantitative and Qualitative Analysis

Our quantitative analysis is based on counting the number of big refactorings performed by students in each project scheme. Refactorings that consists of multiple smaller refactorings, and solve a bigger architecture or design level problem, are considering as big refactorings. The code smells indicating the need for big refactorings were instances of *god classes*, *divergent change*, *shotgun surgery*, *feature envy*, *inappropriate intimacy* and *middleman*. Some of the big refactorings performed were *Extract Hierarchy*, *Separate Domain from Presentation,* and using the *MVC Pattern*.

We compared the average number of big refactorings performed by students working on Project ER to those working on Project RE. It was seen that students of Project ER performed 70% more big refactorings than students of Project RE. Therefore, we deduce that the Project ER scheme results in better quality refactoring.

A qualitative analysis of our work through interviews and meetings with students and general observations by the course instructor, teaching assistant and external evaluator reveals that the primary reason Project ER students had a better understanding of the project was their ability to comprehend the structure of the code during the enhancement phase before moving on to code refactoring. Their better quality of work was entirely due to the opportunity to enhance existing code whereas the Project RE students were seen to gain little understanding by only conducting code analysis using tools or manually. It can be inferred that the best manual analysis is only possible while actually working on enhancing given code. The Project ER students gained more experience by working on the code whereas the Project RE students had to rely on basic code metrics for guiding their refactoring efforts. Since the Project ER provided the opportunity for students to better understand the code, therefore, they were able to produce better results by performing better quality of refactoring. Responses from students from the ER group were indicative of higher satisfaction after refactoring and they were better able to analyze their code for the possibility and opportunities of using relevant design patterns. On the other hand, the efforts by students to apply design patterns were forceful rather than intuitive. Overall, the main reason why Project ER is a better scheme to enhance students learning and makes them perform better quality refactoring is that it helps them gain insight into the core architecture of the project by enhancing first, thereby gaining experience of the issues and code smells.

4.5 Industry Expert's Analysis

As mentioned earlier, an expert from the software industry was invited during the presentations of the students work. It was overall noted by the expert that the Project ER groups performed better than the Project RE groups; 80% students from Project ER were rated above average as compared to only 25% students from Project RE who got this rating. It was also observed by the expert that the Project ER group members had a better understanding of the problems in their code, and they performed refactorings that had a stronger impact. This observation corroborates with the quantitative analysis of results from students' code submissions.

The chart in Figure 2 shows the marks obtained in the final evaluation by the student groups working on the two different project schemes, as assigned by the external evaluator. The observation from this chart is that the students working on Project ER have outperformed those working on Project RE. This answers *RQ1: What is the better scheme to teach refactoring? Should students perform refactoring first or enhancement first?* Since the performance of students working on Project ER is better, we can say with confidence that the better approach to teaching refactoring would be to make the students perform enhancements before refactoring activities.

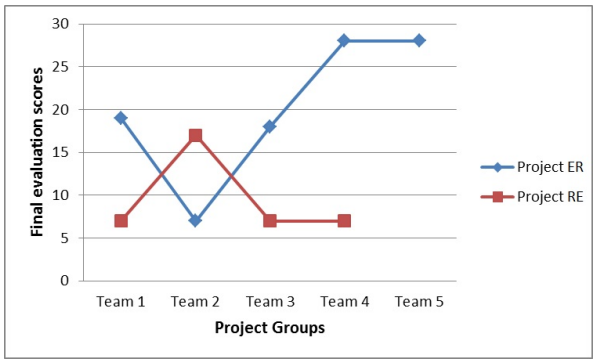

Figure 2. Final evaluation scores of student groups

4.6 Surprises

For the students who were working on Project RE (*pre-enhancement refactoring* scheme), it was assumed or expected that they would write clean code during their enhancement phase after having refactored their code, however, that was not the case. Similarly, 73% students thought that if you refactor before enhancement, the refactoring will take longer, but the enhancement will take less time. However, it turned out that refactoring did not decrease code enhancement time.

4.7 Summary of Results

In short, we found the following answers to our research questions from this experiment:

RQ1. What is the better scheme to teach refactoring? Should students perform refactoring first or enhancement first?

The students who performed enhancement first were better able to understand the structure of code, and it was easier for them to detect potential areas to refactor. Students who refactored before enhancement performed mostly superficial refactorings like *renaming*, *removing comments*, *removing dead code* etc., and the code enhancement phase was not made much easier. Also, the overall performance of students working on *post-enhancement refactoring* scheme was better, as measured in the final evaluations. Therefore, *post-enhancement refactoring* is the better scheme to teach refactoring effectively.

RQ2. How to make students realize the benefit of refactoring?

Both the schemes to carry out refactoring were equally effective in increasing the value perception towards refactoring. However, the *post-enhancement refactoring* scheme resulted in better quality refactorings.

RQ3. Which student group is more satisfied?

Project ER students were slightly more satisfied with their scheme of work because they found it easier to analyze and refactor code after performing enhancement.

4.8 Threats to Validity

4.8.1 Internal Threats

Our claim regarding the success of the Project ER scheme to enable students to better understand the need, application and benefits of refactoring is measured only through their performance in the project and their feedback. It is possible that some confounding factors may have promoted to the success of Project ER scheme, including the previous experience and expertise of the students. The results of the survey may not be totally accurate because a few students were unable to respond. It is also noteworthy that three out of four teams in Project RE were comprised of two members only, whereas two out of five teams in Project ER were comprised of two members. Although the evaluation was done keeping the size of the teams in mind, however, we cannot ignore the possibility of team size affecting the students' performance in the project. Even though the project started out with an equal number of teams in both the project schemes, and an equal number of students in each team, however, when some students opted out of the course once the project started, it was too late to change the grouping structure to balance the number of students in each project scheme. The sample size of our experiment is 23 students only, which might affect the validity of our results.

4.8.2 External Threats

We think that our experiment might produce different results and different user feedback if the project domain and technology platform are changed. It is possible that switching a real life industrial example with an open source project having a familiar domain or with a project that the students have already worked upon in the past would produce different results. Other factors such as the timespan given to refactor code, the quality of lectures, and the students' technical skills might affect the validity of our results.

5. CONCLUSIONS AND FUTURE WORK

In the end, we can conclude that both project schemes were comparable in terms of increasing the students' value perception towards refactoring, however, the *post-enhancement refactoring* resulted in an overall better quality of refactorings performed and better final code structure. It was concluded that the student groups who performed *post-enhancement refactoring* performed better in project assessments, because of their better understanding of the problems in code. It was also deduced that self-written code is better refactored than foreign code.

In terms of the quality of refactorings done, we can say that those who followed *post-enhancement refactoring* scheme performed better quality of refactorings, and their analysis of problems in code was also better. Therefore, we can safely conclude that they had a better learning experience by performing more important and relevant refactorings than the other group. On the other hand, it was observed that the refactoring performed by students on Project RE were superficial, and theirs was a different set of refactorings compared to the other group who knew where the problems were in their own enhancement.

For better statistical results we plan to re-execute the same experiment with more participants. We propose future research directions on conducting refactoring course projects by investigating another project scheme that would involve multiple enhancements and refactoring phases, one after the other, so that the overall effect of refactoring in parallel with enhancement is gained. In future, we might use a project that has a simpler domain, and give flexibility for variation in technology.

6. ACKNOWLEDGEMENTS

We would like to thank Techlogix Pvt. Ltd. for providing us with the source code and Saima Mushtaq for being the evaluator.

7. REFERENCES

[1] Demeyer, S., Van Rysselberghe, F., Girba, T., Ratzinger, J., Marinescu, R., Mens, T., & El-Ramly, M. (2005, September). The LAN-simulation: a refactoring teaching example. In Principles of Software Evolution, Eighth International Workshop on (pp. 123-131). IEEE.

[2] Dibble II, C., & Gestwicki, P. (2014). Refactoring code to increase readability and maintainability: a case study. *Journal of Computing Sciences in Colleges*,30(1), 41-51.

[3] El-Ramly, M. (2006, May). Experience in teaching a software reengineering course. In *Proceedings of the 28th international conference on Software engineering* (pp. 699-702). ACM.

[4] Fowler. M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[5] Kim, M., Zimmermann, T., & Nagappan, N. (2012, November). A field study of refactoring challenges and benefits. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (p. 50). ACM.

[6] Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., & Succi, G. (2008). A case study on the impact of refactoring on quality and productivity in an agile team. In *Balancing Agility and Formalism in Software Engineering* (pp. 252-266). Springer Berlin Heidelberg.

[7] Murphy-Hill, E., Parnin, C., & Black, A. P. (2012). How we refactor, and how we know it. *Software Engineering, IEEE Transactions on*, 38(1), 5-18.

[8] Smith, S., Stoecklin, S., and Serino, C. "An innovative approach to teaching refactoring." *ACM SIGCSE Bulletin* 38.1 (2006): 349-353.

[9] Van Emden, E., & Moonen, L. (2002). Java quality assurance by detecting code smells. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on* (pp. 97-106). IEEE.