RANKINGTHEREFACTORING TECHNIQUES BASED ON THE INTERNAL QUALITY ATTRIBUTES

Sultan Alshehri and Luigi Benedicenti

Department of Engineering, Software Engineering, University of Regina, Canada

ABSTRACT

The analytic hierarchy process (AHP) has been applied in many fields and especially to complex engineering problems and applications. The AHP is capable of structuring decision problems and finding mathematically determined judgments built on knowledge and experience. This suggests that AHP should prove useful in agile software development where complex decisions occur routinely. In this paper, the AHP is used to rank the refactoring techniques based on the internal code quality attributes. XP encourages applying the refactoring where the code smells bad. However, refactoring may consume more time and efforts.So, to maximize the benefits of the refactoring in less time and effort, AHP has been applied to achieve this purpose. It was found that ranking the refactoring techniques helped the XP team to focus on the technique that improve the code and the XP development process in general.

KEYWORDS

Extreme programming, *Refactoring; Refactproing techqinces; Analytic Hierarchy Process.*

1. INTRODUCTION

Code refactoring is the process of improving the design of the existing code by changing its internal structure without changing its external behavior [1]. It is a core activity of the XP development cycle to improve the design of the software and reduce the effort and cost of coding and testing. Most of the current studies focus on the following issues:

1.1 Guidelines in the refactoring process

Mens and Tourwe [2] explained the refactoring steps in detail, which can be summarized as follows:

- Identifying the part of the software that should be refactored.
- Deciding which refactoring methods should be applied
- Applying the refactoring
- Assessing the effects of the applied refactoring methods on the code quality attributes.

Kataoka et al. [3] provided a 3-step model: "identification of refactoring candidates, validation of refactoring effects, and application of refactoring"[4]. Other researcher-provided similar processes can be found in [5,6].

1.2 Issues regarding refactoring tools

Maticorna and Perez [7] presented the refactoring characterization and showed how it can be used as a tool to compare different refactoring definitions, including refactoring catalogs. Also, the authors tackled various refactoring issues, such as design and languages, scope, actions, and application on scheduling, which can each be a good starting point for the builders of the refactoring tools. Murphy-Hill et al. [8] performed an empirical study comparing four methods that were used to gather refactoring data to help in building a powerful refactoring tool. Simmonds and Mens [9] compared four software-refactoring tools: SmalltalkWorks 7.0, Eclipse, Guru, and Together ControlCenter 6.0. They provided detailed results that show the strengths and weakness of each tool. Brunelet al. [10] investigated the accuracy of refactoring tools by analyzing seven open-source java systems (MegaMek, JasperReports, Antlr, Tyrant, PDFBox, Velocity, HSQLDB).Roberts et al. [11] discussed the technical requirements and practical criteria for the refactoring tools. They emphasize that speed and integration are the most practical criteria. Also, the accuracy and the ability to search across the entire program are the most technical requirements.

1.3 Identification of code smells to locate possible refactoring

Sandalski et al. [12] used an intelligent assistant and a refactoring agent to analyze the refactoring architecture and assess the existing code to highlight the portions of code that needed to be refactored and to provide options for the methods.Advani et al. [13] conducted an empirical study using open sources Java software to identify the area of complexity across the systems when refactoring was being applied. Also, they discovered where the refactoring effort was being made. They also created a way for developers to decide how to allocate the testing effort.Hayashi et al. [14] proposed a technique using plug-ins for Eclipse to guide the developer on how and where to do refactoring using the histories of program modification. This technique answered three main questions: where to refactor? Which suitable refactoring method should be used? When should refactoring should apply? Bryton et al. [15] proposed a model called Binary Logistic Regression (BLR) to detect the code smell particularly Long Method objectively.

1.4 The impact of refactoring on the code quality attributes

Many studies; as they will are shown in section 5 have investigated the impact of the refactoring on the code quality attributes. Yet, it is difficult and time consuming to apply all the possible refactoring techniques during the iteration of the development. Therefore, this paper will provide a way of ranking these techniques that can improve the code quality and transfer knowledge to the development team.

2. THE ANALYTICAL HIERARCHY PROCESS

AHP is a systematic approach for problems that involve the consideration of multiple criteria in a hierarchical model. AHP reflects human thinking by grouping the elements of a problem requiring complex and multi-aspect decisions [16]. The approach was developed by Thomas Saaty as a means of finding an effective and powerful methodology that can deal with complex decision-making problems [17]. AHP comprises the following steps: 1) Structure the hierarchy model for the problem by breaking it down into a hierarchy of interrelated decision elements. 2) Define the criteria or factors and construct a pairwise comparison matrix for them; each criterion on the same level is compared with other criteria in respect of their importance to the main goal. 3) Construct a pairwise comparison matrix for alternatives with respect to each objective in separate matrices. 4) Check the consistency of the judgment errors by calculating the consistency ratio. 5) Calculate the weighted average rating for each decision alternative and choose the one with the highest score. More details on the method, including a step-by-step example calculation, are found in [16].Saaty [18] developed a numerical scale for assigning the weight for criteria or alternative by giving a value between 1 (equal importance) and 9 (extreme importance), see table 1.

Table 1. AHP Numerical Scale Developed by Saaty [18].

Scale	**Numerical Rating**	**Reciprocal**
Equal importance	1	1
Moderate importance of one over other	3	1/3
Very strong or demonstrated importance	7	1/7
Extreme importance	9	1/9
Intermediate values	2,4,6,8	1/2, 1/4, 1/6, 1/8

3. REFACTORING TECHNIQUES

Fowler [1] assures that the refactoring helps developers to program faster, to find bugs and to improve the software design. So, he defined more than 70 different refactoring patterns and organized them into six categories: composing methods, moving features between objects, organizing data, simplifying conditional expressions, making methods calls simpler, and dealing with generalization. Each of the refactoring patterns has a specific purpose and effect over the quality attributes. However, projects can have different priorities in terms of the quality attributes. Using one or more of the refactoring methods variously improves the code and the design of the software. So, it is essential to allocate the team’s efforts to the most important quality attributes in order to maximize the value expected from the system. It is often unclear to software designers how to use refactoring methods to improve particular quality attributes [19]. The task of selecting the refactoring patterns is time-consuming and can create a conflict between the programmers’ opinions. Pivet et al. [20] introduced the AHP for three techniques and compared them based on the use of these patterns, particularly in terms of simplicity, reusability, and comprehensibility. These patterns would be more beneficial if we could investigate more patterns and rank them based on their influence on the code instead of their capabilities and uses. Therefore, in this section, we focused on ranking refactoring patterns based on their effects on the code quality rather than their characteristics of uses. Also, we have chosen eight refactoring patterns from four different categories proposed by Fowler to show the importance of these refactoring techniques using AHP. The following patterns were selected:

- Extract Method, Inline Method, and Inline Temp Method from "Composing Methods" category.
- Extract Class, Inline Class, and Move Method from "Moving Features Between Objects" category.
- Rename Method from "Making Method Calls Simpler" category.
- Pull Up Method from "Dealing with Generalization" category.

4. CASE STUDIES SETUP

The research questions, propositions, units of analysis,data collections and sources, and designing the case studies are presented in this section.

4.1 Research questions and propositions

The primary objective in the refactoring practice is to investigate how AHP can be used in ranking the internal and external code quality attributes. The following research questions provided a focus for our case study investigation:

- How important is it to practice the refactoring using AHP?
- How can AHP rank the refactoring methods based on specific criteria?
- How can AHP affect the communication among the developers?
- How can AHP help in saving the developers' time while refactoring?

The methodology used in this study is four case studies: two case studies in an academic environment and two case studies in industrial environments with embedded units of analysis. The study propositions are outlined below:

- AHP captures important criteria and alternatives that need to be considered when refactoring.
- AHP facilitates the process of ranking and selection in refactoring.
- AHP involves an informative discussion and improves the communication among the developers.
- AHP provides a map to focus on the most important refactoring methods that increase the code quality.
- AHP resolves conflicting opinions among the developers when practicing the refactoring and trying to find the smell code.

4.2 Unit of analysis

According to [21] the unit of the analysis should be derived from the main research questions of the study. So, the main focus of this study is to rank the refactoring pattern based on internal qualities attributes. So, the ranking and the process of evaluation are units of analysis for this study. Also, the developers' view of how AHP benefits each XP practice is another unit analysis. As result, this work is designed as multiple cases (embedded) with multiple units of analysis.

4.3 Data collection and sources

In the beginning of the applying AHP to the refactoring practice, we propose the criteria that we want to investigate in order to examine the AHP tool's ability and benefits. This data was collected from literature review and previous studies. To increase the validity of this study, data triangulation was obtained. The data sources in this study were:

- Archival records, such as a study plan from the graduate students.
- Questionnaires given to the participants when developing the XP project.
- Questionnaires given to experts from industry.
- Open-ended interviews with the participants.
- Feedback from the customer.

However, the most important data source of this study was an XP project conducted at the University of Regina in a software design class in fall 2012. In addition, three companies initially participated in evaluating some of the XP practices and based on proposed criteria that affect the practice. Later on, two of the three companies involved in validating the results.

4.4 Designing the case studies

The educational case studies were performed as part of a course in the Advanced Software Design Class for graduate students taught in Fall 2012 at the University of Regina. The participants were 12 master’s students and a client from a local company in Regina. Participants have various levels of programming experience and a good familiarity with XP and its practices. The Students' background related to the experiment includes several programming languages such as Java, C, C#, and ASP.net. They have implemented projects previously using various software process methodologies. The study was carried out throughout 15 weeks; students were divided into two teams. Both teams were assigned to build a project called “Issue Tracking System” brought by the client along with industrial requirements. It ran in 5 main iterations and by the end of the semester, the whole software requirements were delivered. The students were paired based on their experience and knowledge, but also we had an opportunity to pair some experts with novice and average programmers for the purpose of the study. Participants were given detailed lectures and supporting study materials on extreme programming practices that focused on planning game activities which included writing user stories, prioritizing the stories, estimating process parameters, and demonstrating developers’ commitments. The students were not new to the concept of XP, but they gained more knowledge and foundation specifically in the iteration plan, release planning and prioritizing the user stories. In addition, the students were exposed to the AHP methodology and learned the processes necessary to conduct the pairwise comparisons and to do the calculations. Several papers and different materials about the AHP and user stories were given to the students to train them and increase their skills in implementing the methodology. In addition, a survey was distributed among students to get further information about their personal experiences and knowledge.

The researcher have visited three companies (two companies in Regina, and one in Calgary; both in Canada) several times and met with the developers and team leaders to explain the purpose of the study and to collect the data and feedback from the real industries. To preserve their anonymities, A, B, and C replace the companies’ real names. All the companies are familiar with XP concept and currently practicing the refactoring during their development. In this study, eighteen experts have used their knowledge and average of 10 years experience to evaluate the proposed pair alternatives using the AHP.

5. AHP USEINREFACTORING

The AHP method can be used in the XP development team to rank the refactoring techniques based on the internal code quality attributes. In the following sections, the AHP evaluation, structure and process are presented.

5.1 Background

Before applying the AHP to the refactoring, we will highlight some of the previous studies that already have investigated the impact of the refactoring on the internal code quality attributes as follow:

Moser et al. [22] conducted a case study to assess the refactoring impacts in an industrial environment. They found that refactoring improved software quality attributes such as coupling, cohesion, and response for a class. Stroggylos and Spinellis [23] analyzed source code version control system logs from four open source software systems to detect changes that occurred after refactoring and to examine the refactoring's impact on the software metrics process. The metrics examined include coupling, cohesion, and number of methods. Bois et al. [24] analyzed how refactoring changes the coupling and cohesion characteristics. These refactoring methods were used: Move Method, Replace Method with Method Object, Replace Data Value with Object and Extract Class. The study led to an improvement in the code quality. Moser et al. [25] conducted a case study in agile environment. The quality metrics selected for this study were complexity, coupling, cohesion, number of methods per class, response of a class, depth of inheritance tree, and number of children. They used a methodology proposed in their research that led them to the conclusion that the refactoring could significantly improve the internal measures for reusability of object-oriented classes written in Java. Ratzinger et al. [26] analyzed the history of a large industrial system during 15 months and showed how refactoring can reduce the change couplings and enhance the software evolvability. Stroulia and Kapoor [27] investigated how the refactoring could improve the design and code quality. They found that the average LOC, the average the number of statements, the number of methods, and the number of collaborators were decreased in the individual system classes. Kataoka et al. [28] found that refactoring techniques such as extract method and Extract Class reduce the coupling in the code and improve the maintainability of the system. Zhao and Hayes [29] introduced a rank-based software using a measure-driven refactoring decision to support the development team's decisions of where to apply resources when they did refactoring. They presented two case studies that examined the approach that identifies which classes and packages need to be refactored based on static measures such as code size, coupling, and complexity. Geppert et al. [30] found the defect rates and change efforts decreased significantly.

5.2 Proposed criteria for ranking the refactoring

To rank the refactoring techniques, it is necessary to identify the quality attributes that are more valuable to the development team or the organization. Each project can have a different set of criteria and alternative refactoring methods in order to be ranked and evaluated. In this paper, we have chosen four internal quality attributes as the core criteria for ranking the refactoring techniques:

- Cohesion: each system component does one thing and does it well.
- Coupling: the degree to which each system component relies on another component.
- Complexity: the degree of connectivity between elements of a design unit.
- Code size: most common technical sizing method is number of Lines Of Code (#LOC) per technology, number of files, functions, classes, tables, etc.

5.3 AHP-refactoring structure for the internal attributes

The first step in the analytic hierarchy process is to structure the problem as a hierarchy that includes three levels. The top level is the main objective: ranking the refactoring techniques; the second level is the criteria: complexity, cohesion, coupling, and code size; the third level is the alternatives: Extract Method, Inline Method, InlineTemp Method, Extract Class, Inline Class, Move Method, Pull Up Method, and Rename Method. Figure 1 illustrates the AHP structure for the problem.

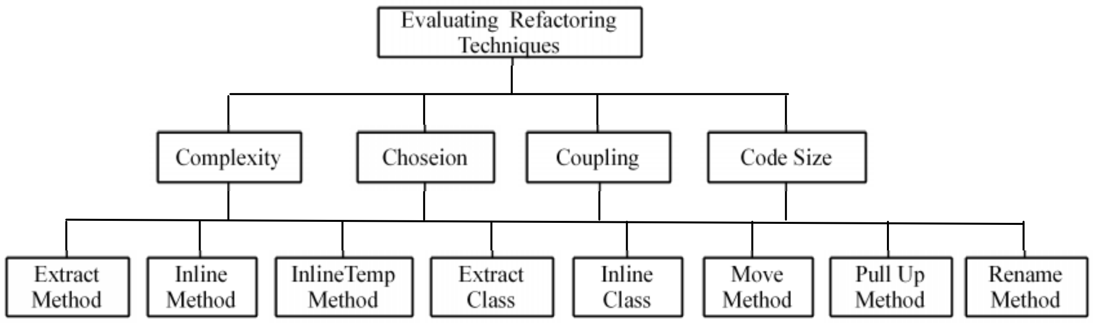

Figure 1. AHP Structure for the refactoring techniques based on the internal attributes

5.4 Refactoring pairwise comparison process

All the participants had to apply the refactoring patterns to a real project to see the real impact on their code. Then they were required to evaluate the refactoring patterns based on certain criteria. For this purpose, sheets of paper with appropriate AHP tables were handed to the all participants in order to save time and facilitate the process of comparison. The participants compared the criteria using the Saaty scale from 1-9. The participants were asked these questions:

- Which is more important: complexity or cohesion and by how much?
- Which is more important: complexity or coupling and by how much?
- Which is more important: complexity or code size and by how much?
- Which is more important: cohesion or coupling and by how much?
- Which is more important: cohesion or code size and by how much?
- Which is more important: coupling or code size and by how much?

After finishing the criteria comparisons, the participants had to evaluate all the refactoring techniques based on each criterion every time. Example follows:

- In term of reducing the complexity, which is more important Extract Method or Inline Method and by how much?

Similarly, all the following comparisons were conducted based on each criterion:

- (Extract Method **X** Inline Method), (Extract Method **X** Inline Temp Method), (Extract Method **X** Extract Class), (Extract Method **X** Inline Class), (Extract Method **X** Move Method), (Extract Method **X** Inline Pull Up Method) (Extract Method **X** Rename Method).
- (Inline Method **X** Inline Temp Method), (Inline Method **X** Extract Class), (Inline Method **X** Inline Class), (Inline Method **X** Move Method) (Inline Method **X** Inline Method), (Inline Method **X** Inline Pull Up Method), (Inline Method **X** Rename Method).
- (Inline Temp Method **X** Extract Class), (Inline Temp Method **X** Inline Class), (Inline Temp Method **X** Move Method), (Inline Temp Method **X** Pull Up Method), (Inline Temp Method **X** Rename Method).
- (Extract Class **X** Inline Class), (Extract Class **X** Move Method), (Extract Class **X** Pull Up Method), (Extract Class **X** Rename Method).
- (Inline Class **X** Move Method), (Inline Class **X** Pull Up Method), (Inline Class **X** Rename Method).
- (Move Method **X** Pull Up Method) (Move Method **X** Rename Method).
- (Pull Up Method**X** Rename Method).

The same questions and comparisons repeated until the participants evaluated all refactoring techniques based on each criterion.

6. AHP EVALUATIONRESULTS

6.1 Educational case studies

For Team 1, the rankings of the refactoring techniques based on all criteria, i.e. complexity, cohesion, coupling and code size, are summarized as follows. First: Extract Class (17.61); Second: Extract Method (15.37); Third: Inline Class (15.35); Fourth: Pull Up Method (13.71); Fifth: Move Method (11.55); Sixth: Inline Temp method (10.96); Seventh: Inline Method (10.17); Eighth (5.28). Table 2 summarizes the results.

Figure 2 shows the importance of each criterion as follows: coupling (29.29), cohesion (29.25), code size (22.56), and complexity (18.90).

Table 2. Ranking the refactoring techniques by Team 1

Refactoring Techniques	All
Extract Class	17.61 %
Extract Method	15.37 %
Inline Class	15.35 %
Pull Up Method	13.71 %
Move Method	11.55 %
Inline Temp Method	10.96 %
Inline Method	10.17 %
Rename Method	5.28 %

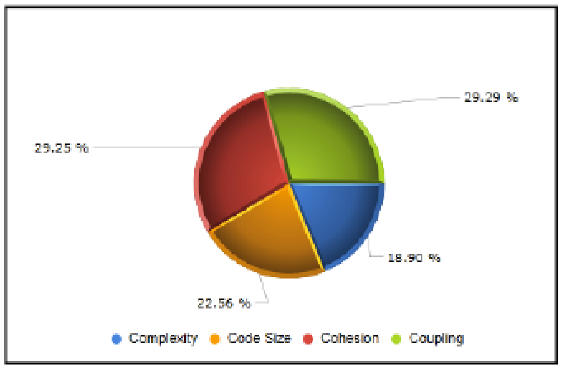

Figure 2. Importance of the internal criteria for the Team 1

Team 2's rankings of the prioritization techniques is summarized as follows: First: Inline Class (19.31); Second: Inline Method (15.65); Third: Extract Class (13.42); Fourth: Extract Method (13.35); Fifth: Move Method (12.19); Sixth: Inline Temp Method (9.69); Seventh: Inline Method (9.56); Eighth: Rename Method (6.83). Table 3 summarizes the results.

Figure 3 shows the importance of each criterion as follows: coupling (33.61), cohesion (32.37), complexity (27.10), and code size (6.93).

Table 3. Ranking the refactoring techniques by Team 2

Refactoring Techniques	All
Inline Class	19.31 %
Inline Method	15.65 %
Extract Class	13.42 %
Extract Method	13.35 %
Move Method	12.19 %
Inline Temp Method	9.69 %
Pull Up Method	9.56 %
Rename Method	6.83 %

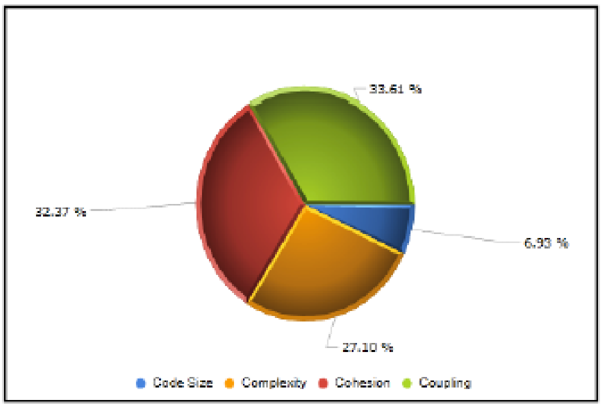

Figure 3. Importance of the internal criteria for Team 2

6.1.1 Observations:

1. Considering all the criteria, the Extract Class technique was ranked the highest by Team 1, while Team 2 ranked it in the third position. Team 2 ranked the Inline Class in the highest position while Team 1 ranked it in the third position.
2. Both Extract Class and Inline Class are categorized by Fowler as “Moving Features Between Objects.”
3. Team 1 ranked Extract Method in the second position, and Team 2 ranked the Inline Method in the second position. Both Extract Method and Inline Method are categorized by Fowler as “Composing Methods”.
4. Rename Method was ranked in the last position by the two teams.
5. Cohesion and coupling quality attributes were of the highest concern for both teams.
6. If we look at the refactoring techniques considering each criterion individually, we can see both teams ranked the Extract Method in the top position in the complexity attributes. Also, both teams ranked the Inline Class in the top position in the cohesion attributes. See tables 4 and 5.
7. For the coupling quality attribute, Team 1 ranked the Extract Method in the highest position, while Team 2 ranked the Inline Class at the top. See tables 4 and 5.
8. For the code size quality attribute, Team 1 ranked the Extract Class in the highest position, while Team 2 ranked the Inline Method at the top. See tables 4 and 5.

Table 4. Refactoring techniques based on each internal criterion by Team 1

Refactoring Techniques	Complexity	Refactoring Techniques	Coupling	Refactoring Techniques	Chohesion	Refactoring Techniques	Code Size
Extract Method	19.30 %	Extract Method	18.54 %	Inline Class	18.17 %	Extract Class	21.50 %
Extract Class	19.29 %	Extract Class	16.62 %	Pull Up Method	16.41 %	Inline Class	17.24 %
Inline Class	15.00 %	Pull Up Method	12.68 %	Extract Class	15.02 %	Inline Temp Method	13.09 %
Pull Up Method	14.70 %	Move Method	12.44 %	Inline Method	11.83 %	Extract Method	12.64 %
Move Method	13.16 %	Inline Class	11.43 %	Extract Method	11.65 %	Inline Method	11.16 %
Inline Method	7.76 %	Inline Temp Method	11.30 %	Inline Temp Method	11.56 %	Move Method	10.32 %
Inline Temp Method	7.11 %	Inline Method	9.33 %	Move Method	10.50 %	Pull Up Method	10.20 %
Rename Method	3.68 %	Rename Method	7.66 %	Rename Method	4.87 %	Rename Method	3.84 %

Table 5. Refactoring techniques based on each internal criterion by Team 2

Refactoring Techniques	Complexity	Refactoring Techniques	Coupling	Refactoring Techniques	Chohesion	Refactoring Techniques	Code Size
Extract Method	17.88 %	Inline Class	21.91 %	Inline Class	22.47 %	Inline Method	17.23 %
Extract Class	17.26 %	Inline Method	21.68 %	Inline Method	16.39 %	Inline Temp Method	14.89 %
Inline Class	14.97 %	Extract Method	14.37 %	Extract Class	15.40 %	Inline Class	14.44 %
Pull Up Method	13.78 %	Move Method	12.12 %	Move Method	13.47 %	Pull Up Method	13.74 %
Move Method	11.40 %	Inline Temp Method	10.62 %	Inline Temp Method	10.27 %	Extract Method	11.11 %
Rename Method	9.45 %	Extract Class	8.59 %	Pull Up Method	9.08 %	Move Method	10.79 %
Inline Method	8.43 %	Rename Method	5.97 %	Extract Method	8.13 %	Extract Class	10.31 %
Inline Temp Method	6.83 %	Pull Up Method	4.74 %	Rename Method	4.79 %	Rename Method	7.48 %

6.2 Industrial case studies

The rankings for the prioritization of techniques by company Aare summarized as follows: First: Extract Method (16.21); Second: Pull Up Method (15.94); Third: Inline Class (15.8); Fourth: Extract Class (13.7); Fifth: Rename Method (11.01); Sixth: Inline Method (10.9); Seventh: Move Method (8.62); Eighth: Inline Temp Method (8.51). Table 7 summarizes the results.Figure 4 shows the importance of each criterion as follows: code size (37.93), cohesion (30.72), coupling (16.21), and complexity (15.14).

Table 7. Ranking the refactoring techniques by company A

Refactoring Techniques	All
Extract Method	16.21 %
Pull Up Method	15.94 %
Inline Class	15.8 %
Extract Class	13.7 %
Rename Method	11.01 %
Inline Method	10.9 %
Move Method	8.62 %
Inline Temp Method	8.51 %

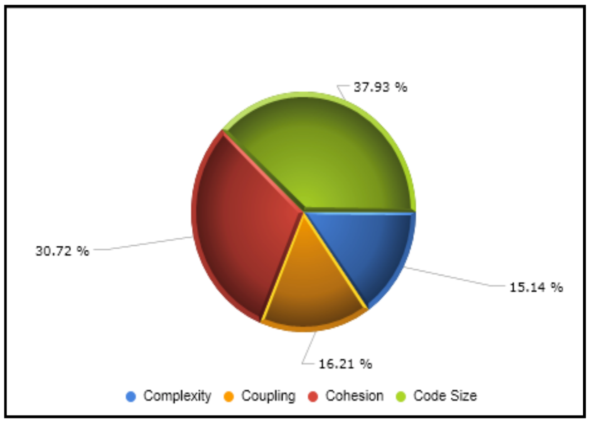

Figure 4. Importance of the internal criteria for refactoring company A

The rankings for the prioritization of techniques by company Bare summarized as follows: First: Extract Class (24); Second: Extract Method (19.99); Third: Move Method (12.44); Fourth: Inline Class (10.25); Fifth: Pull Up Method (9.79); Sixth: Inline Method (8.83); Seventh: Rename Method (8.31); Eighth: Inline Temp Method (6.39). Table 8 summarizes the results.

Figure 5 shows the importance of each criterion as follows: cohesion (41.75), coupling (30.34), complexity (14.83), and code size (13.08).

Table 8. Ranking the refactoring techniques by company B

Refactoring Techniques	All
Extract Class	24 %
Extract Method	19.99 %
Move Method	12.44 %
Inline Class	10.25 %
Pull Up Method	9.79 %
Inline Method	8.83 %
Rename Method	8.31 %
Inline Temp Method	6.39 %

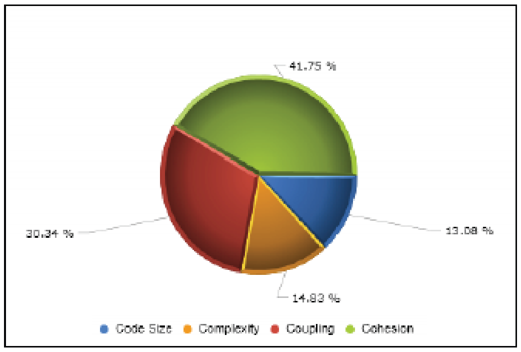

Figure 5. Importance of the internal criteria for company B

The rankings for the prioritization of techniques by company Care summarized as follows: First: Extract Class (18.46); Second: Inline Class (17.98); Third: Extract Method (14.19); Fourth: Inline Method (13.89); Fifth: Inline Temp Method (10.58); Sixth: Move Method (10.4); Seventh: Pull Up Method (9.21); Eighth: Rename Method (5.29). Table 9 summarizes the results.

Figure 6 shows the importance of each criterion as follows: cohesion (38.37), coupling (38.76), code size (12.38), and complexity (10.49).

Table 9. Ranking the refactoring techniques by company C

Refactoring Techniques	All
Extract Class	18.46 %
Inline Class	17.98 %
Extract Method	14.19 %
Inline Method	13.89 %
Inline Temp Method	10.58 %
Move Method	10.4 %
Pull Up Method	9.21 %
Rename Method	5.29 %

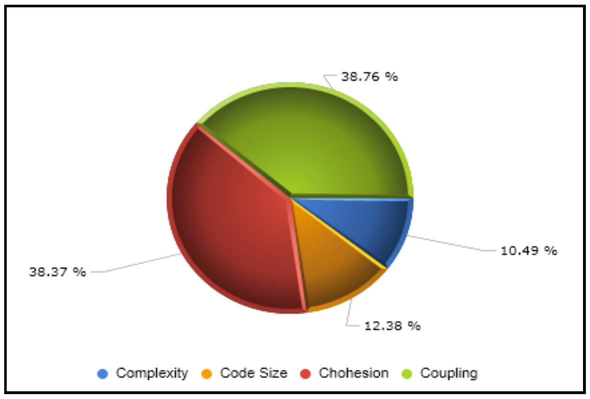

Figure 6. Importance of the internal criteria for the refactoring by company C

6.2.1 Observations:

1. Considering all the criteria together, the Extract Class technique was ranked highest by companies B and C while company A ranked it in the fourth position. Company A ranked the Extract Method in the highest position.

2. The second position for each company was different. The second-highest position in A was Pull Up Method, in company B, it was Extract Method, and in company C it was Inline Class.
3. Inline Temp Method was ranked in the last position by companies A and B, while company C ranked the Rename Method in the last position.
4. Cohesion and coupling quality attributes were of the highest concern for companies B and C, while company A was most concerned with the code size and cohesion.
5. If we look at the refactoring techniques and consider each criterion individually as it is shown in tables 10, 11, and 12, we can see companies A and B ranked the Extract Class in the top position in the complexity attributes, while company C ranked the Extract Method at the top.
6. For the cohesion quality attribute, companies B and C ranked the Extract Class in the highest position, while company A ranked the Extract Method at the top.
7. For the coupling quality attribute, companies A, B, and C all ranked the Extract Class in the highest position.
8. For the code size, all the companies ranked the refactoring techniques differently. Companies A, B and C ranked the Pull Up Method, Inline Method, and Inline Class in the highest position, respectively.

Table 10. Refactoring techniques based on each internal criterion by company A

Refactoring Techniques	Complexity	Refactoring Techniques	Coupling	Refactoring Techniques	Chohesion	Refactoring Techniques	Code Size
Extract Class	22.74 %	Extract Class	17.16 %	Extract Method	22.22 %	Pull Up Method	15.71 %
Extract Method	15.21 %	Pull Up Method	16.43 %	Inline Class	21.05 %	Inline Class	15.05 %
Pull Up Method	13.19 %	Extract Method	13.47 %	Rename Method	14.88 %	Extract Method	14.4 %
Inline Method	11.02 %	Rename Method	13.35 %	Extract Class	12.58 %	Inline Method	13.28 %
Move Method	10.49 %	Inline Class	11.98 %	Pull Up Method	8.46 %	Move Method	12.78 %
Inline Class	10.1 %	Inline Method	9.62 %	Inline Method	8.05 %	Inline Temp Method	12.25 %
Inline Temp Method	8.77 %	Inline Temp Method	8.38 %	Inline Temp Method	6.83 %	Extract Class	9.41 %
Rename Method	8.47 %	Move Method	8.59 %	Move Method	5.93 %	Rename Method	7.12 %

Table 11. Refactoring techniques based on each internal criterion by company B

Refactoring Techniques	Complexity	Refactoring Techniques	Coupling	Refactoring Techniques	Chohesion	Refactoring Techniques	Code Size
Extract Class	28.73 %	Extract Class	28.85 %	Extract Class	24.23 %	Inline Method	18.41 %
Extract Method	21.4 %	Extract Method	23.53 %	Extract Method	20.19 %	Inline Class	18.38 %
Move Method	12.39 %	Move Method	11.92 %	Move Method	13.83 %	Extract Method	13.24 %
Pull Up Method	10.28 %	Pull Up Method	9.59 %	Inline Class	10.32 %	Extract Class	12.81 %
Rename Method	9.39 %	Rename Method	8.4 %	Pull Up Method	10.26 %	Inline Temp Method	11.57 %
Inline Method	7.03 %	Inline Class	6.55 %	Rename Method	8.39 %	Move Method	9.87 %
Inline Class	6.42 %	Inline Method	5.79 %	Inline Method	7.28 %	Pull Up Method	8.55 %
Inline Temp Method	4.35 %	Inline Temp Method	5.37 %	Inline Temp Method	5.5 %	Rename Method	7.17 %

Table 12.Refactoring techniques based on each internal criterion by company C

Refactoring Techniques	Complexity	Refactoring Techniques	Coupling	Refactoring Techniques	Chohesion	Refactoring Techniques	Code Size
Extract Method	19.96%	Extract Class	20.46 %	Extract Class	18.8 %	Inline Class	26 %
Inline Class	16.98 %	Inline Class	15.93 %	Inline Class	18.28 %	Inline Temp Method	15.16 %
Extract Class	13.7 %	Extract Method	15.41 %	Inline Method	14.85 %	Extract Class	14.31 %
Inline Method	13.4 %	Inline Method	13.4 %	Extract Method	13 %	Inline Method	12.23 %
Move Method	11.12 %	Pull Up Method	11.39 %	Inline Temp Method	11.82 %	Pull Up Method	9.94 %
Inline Temp Method	10.65 %	Move Method	11.17 %	Move Method	9.74 %	Move Method	9.32 %
Pull Up Method	9.61 %	Inline Temp Method	8.08 %	Pull Up Method	6.91 %	Extract Method	8.08 %
Rename Method	4.58 %	Rename Method	4.16 %	Rename Method	6.6 %	Rename Method	5.02 %

6.3 Impact of refactoring on the quality attributes

Tables 13 and 14 show the impact of the refactoring on the internal quality attributes reported by Team 1 and Team 2.

Table 13. Refactoring impact on the internal attributes by Team 1

	Complexity	Cohesion	Coupling	Code Size
Extract Method	-	+	-	-
Inline Method	0	+	-	-
InlineTemp Method	0	0	0	0
Extract Class	0	0	0	0
Inline Class	+	+	-	-
Move Method	+	+	-	-
Pull Up Method	-	+	-	-
Rename Method	0	0	0	0

Table 14. Refactoring impact on the internal attributes by Team 2

	Complexity	Cohesion	Coupling	Code Size
Extract Method	-	+	0	+
Inline Method	+	0	0	-
InlineTemp Method	-	0	0	-
Extract Class	-	-	+	+
Inline Class	+	+	-	-
Move Method	-	-	+	0
Pull Up Method	-	+	+	-
Rename Method	-	0	0	0

7. REFACTORINGVALIDATION

From the previous AHP evaluation conducted by students and experts, we found out that there are three refactoring techniques that have received high rankings: Extract Class, Inline Class, and Extract Method.In this section, we collected technical information from two educational studies (Team 1 and 2) and two industrial studies (Company A and B) to validate the AHP evaluation results obtained previously. (We could not obtain some of the data from company C). The collected information included the following:

- The impact of the proposed refactoring techniques on the internal and external quality attributes.
- How many times each refactoring technique was used for each iteration in each case study.

In addition, the participants were required to count the time spent for the refactoring before and after using AHP.

6.4.9.1 Number of times using the refactoring techniques

Tables 15 and 16 summarize how many times each of the refactoring techniques were used in each iteration by companies A and B.

Table 15 Number of times refactoring was used for each iteration by company A

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Total
Extract Method	27	34	21	49	41	172
Inline Method	5	13	17	8	22	65
InlineTemp Method	0	4	0	7	5	16
Extract Class	18	11	12	15	9	65
Inline Class	18	10	7	11	3	49
Move Method	8	6	0	0	20	34
Pull Up Method	21	17	9	27	13	87
Rename Method	11	0	17	22	5	55

Table 16 Number of times refactoring was used for each iteration by company B

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Total
Extract Method	9	5	2	1	1	18
Inline Method	2	1	0	0	0	3
InlineTemp Method	1	0	0	0	0	1
Extract Class	5	5	3	3	2	18
Inline Class	1	0	0	0	0	1
Move Method	2	0	0	0	0	2
Pull Up Method	7	5	2	2	1	17
Rename Method	4	1	0	0	0	5

7.2 Refactoring impact on the internal quality attributes

Tables17and 18 summarize the internal impacts for each refactoring technique for companies A and B.

Table 17. Refactoring impact on the internal attributes by company A

	Complexity	Cohesion	Coupling	Code Size
Extract Method	-	+	-	+
Inline Method	-	0	+	-
InlineTemp Method	+	-	0	+
Extract Class	-	+	-	-
Inline Class	+	-	-	+
Move Method	-	-	+	-
Pull Up Method	-	0	-	-
Rename Method	-	0	0	0

Table 18. Refactoring impact on the internal attributes by company B

	Complexity	Cohesion	Coupling	Code Size
Extract Method	-	+	-	+
Inline Method	+	-	+	-
InlineTemp Method	+	0	0	-
Extract Class	-	+	-	+
Inline Class	+	0	+	+
Move Method	+	-	0	+
Pull Up Method	-	+	-	-
Rename Method	-	0	0	0

7.3 AHP-refactoring impact on time

Both companies were asked to provide the time spent before and after using the AHP. Table 17 shows that the time was reduced.

Table 19. Refactoring impact on time for company A and B

	Time before AHP	After AHP
Company A	3 Days	2 Days
Company B	12 hours	7 hours

7.4 Observations from the validation

- From the AHP evaluation results, Extract Class and Extract Method were mostly ranked in the top positions in the internal quality attributes. Tables 15 and 16 show that Extract Method and Extract Class are commonly used and havepositive effects (asseen in tables 17 and 18). In Company A, the most refactoring techniques were used: Extract Method (172), Pull Up Method (87), and Extract Class and Inline Method (65). In Company B, the fewest refactoring techniques were used: Extract Method and Extract Class (18), and Pull Up Method (17).
- Extract Class showed a positive impact on all the internal quality attributes in company A. The complexity, coupling and code size were reduced while the cohesion increased. In company B, the complexity and coupling reduced as well, but the code size increased.

The cohesion increased as positive result. For both companies, Extract Method showed a positive impact on all the internal quality attributes, except the code size increased.

- Pull Up Method was used many times by the two companies, even though it was not showing significant results by the AHP evaluation. Also, it showed a positive impact on both internal quality attributes generally.
- After narrowing the use of refactoring techniques and ranking them using AHP, the time for refactoring for company A was reduced from 3 days to 2 days, while the refactoring time for company B was reduced from 12 hours to 7 hours.

8. SEMI-INTERVIEW RESULTS

The semi-interview was conducted after showing the participants the results of the AHP evaluation for the refactoring practices. Some of the results were surprising and others were expected. The interview included open questions to obtain students' general opinions about the AHP, advantages and disadvantage of the using the AHP, and the best experience of the AHP among all the XP practices. As noted previously, the data was collected in the form of handwritten notes during the interviews. These notes were organized in a folder for the sake of easy access and analysis. The questions and answers for the semi-interview are below and the people's names are kept anonymous.

From the interviews, we found very positive feedback from the participants regarding the AHP. The AHP resolved any conflicting opinions and brought each team member's voice to the decision in a practical way. It also empathized with the courage of the team by letting every opinion be heard. The time and the number of the comparisons were the main concerns of the participants. All of them recommended using the AHP in the future when it needs to decide which refactoring should be applied. There were a few additional recommendations as well, such as developing an automated tool to reduce the time for the AHP calculation, adding the mobility features, performing cost and risk analysis, and trying it with other XP areas and studying the outcomes.

9. QUESTIONNAIRES

Questionnaires were given to the participants in order to obtain their perceptions of and experiences with the AHP. The questionnaires were divided into two main parts. The first part contained questions about the AHP as a decision and ranking tool. The second part contained questions regarding the direct benefits of the XP practice and investigated the participants' satisfaction. We used a seven-point Likert scale to reflect the level of acceptability of the AHP tool. The seven-point scale appeared as follows: (1) Totally unacceptable. (2) Unacceptable. (3) Slightly unacceptable. (4) Neutral. (5) Slightly acceptable. (6) Acceptable. (7) Perfectly Acceptable.

Once the participants completed the questionnaire, we calculated the results and presented the total percentage of the acceptability for each statement in the evaluation (questionnaires) in the tables 4, 5, and 6.

The total percentage of the acceptability was calculated as follows:

- The total percentage of acceptability (TPA)
  = The average of the score for each team * 100 / 7.
- The average of the score for each team =
  = The sum of the scores given by the team members / number of the team.

9.1 Acceptability level of AHP as a decision and ranking tool

AHP received positive ratings by the two teams and the three companies on most of the questions. However, the lowest percentage given by all studieswere regarding the time efficiency(59%, 62%, 61%, 57%, 57%). See table 20.

Table 20. Acceptability level of AHP as a decision and ranking tool

	Team 1	Team 2	Company A	Company B	Company C
1-AHP as a Ranking tool in Refactoring **A- Decision Quality:**					
Capturing the needed information	76%	88%	86%	83%	88 %
Clarity of the decision process	88%	86%	94%	94%	90%
Clarity of the criteria involved	81%	76%	88 %	83%	88 %
Clarity of the Alternatives involved	81%	79%	88%	90%	95%
Goodness of the decision structure	86%	90%	98%	88%	71%
B- Practically					
Understandability	83%	88%	98%	90%	85.66
Simplicity	71%	86%	76%	74 %	74 %
Time Efficiency	59%	62%	61%	57%	57%
Reliability	74%	76%	81%	90%	90%

9.2 Acceptability level for the impact of AHP on the development:

The following percentages show the acceptability level for the impact of the AHP on the development:

- First: improving team communication; Team1 (74%), Team2 (93%), company A (93%), company B (93%), and company C (94%).
- Second: creating aninformative discussion and learning opportunities, Team1 (71%),Team2 (88%), company A (86%), company B (95%), and company C (93%).
- Third: clarifying the ranking problem; Team1 (71%), Team2 (90%), company A (86%), company B (93%), and company C (83%).
- Fourth: resolving the conflicting opinions among members; Team1 (64%), Team2 (86%), company A (86%), company B (93%), and company C (83%).
- Fifth: elevating the team performance; Team1 (76%) and Team2 (88%), company A (79%), company B (86%), and company C (not study).

10. VALIDITY

Construct validity, internal validity, external validity and reliability describe common threats to the validity tony performed study [31].”Empirical studies in general and case studies in particular are prone to biases and validity threats that make it difficult to control the quality of the study to generalize its results” [32]. In this section, relevant validity threats are described. A number of possible threats to validity can be identified for this work.

10.1 Construct validity

Construct validity deals with the correct operational measures for the concept being studied and researched. The major construct validity threat to this study is the small number of participants in each case study. This threat was mitigated by using several techniques in order to ensure the validity of the findings, as outlined below.

- Data triangulation: A major strength of case studies is the possibility of using many different sources of evidence [33]. This issue was taken into account through the use of surveys and interviews with different types of participants from different environments with various levels of skills and experience, and through the use of several observations as well as feedback from those involved in the study. By establishing a chain of evidence, we were able to reach a valid conclusion.
- Methodological triangulation: The research methods employed were a combination of a project conducted to serve this purpose, interviews, surveys, AHP result comparisons, and researchers’ notes and observations.
- Member checking: Presenting the results to the people involved in the study is always recommended, especially for qualitative research. This was done by showing the final results to all participants to ensure the accuracy of what was stated and to guard against researcher bias.

10.2 Internal validity

Internal validity is only a concern for an explanatory case study [33]. Internal validity focuses on establishing a causal relationship between students and educational constraints. This issue can be addressed by relating the research questions to the propositions and other data sources providing information regarding the questions.

10.3 External validity

External validity is related to the domain of the study and the possibilities of generalizing the results. To provide external validity to this study, we will need to conduct an additional case study in the industry involving experts and developers, and then observe the similarities and the differences in the findings of both studies. Thus, future work will contribute to accruing external validity.

10.4 Reliability

Reliability deals with the data collection procedure and results. Other researchers should arrive at the same case study findings and conclusions if they follow the same procedure. We addressed this by making the research questions, case study set up, data collection and analysis procedure plan is available for use by other researchers.

11. CONCLUSIONS

After using the AHP to rank the refactoring techniques, it was found to be an important tool that provides a very good vision for developers when they want to apply the refactoring practice to improve the code. Considering the complexity, cohesion, coupling and code size when ranking the refactoring techniques could bring many advantages to the development team such as code enhancing the code in a short time and transferring knowledge to the developers. The Extract Class and Extract Method were the most refactoring techniques have improved the code in our studies. However, the other refactoring techniques have added values to internal code qualities as well. More importantly, though, the AHP has a positive impact on the development process and communication among the team members. For example, the team could mathematically reconcile the conflict of opinions regarding the use of refactoring techniques. The AHP provides a cooperative decision making environment, which could maximize the effectiveness of the developed software.

References

[1] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, “Refactoring: Improving the Design of Existing Code”, Addison-Wesley Professional; 1 edition (July 8, 1999).

[2] Tom Mens, Tom Tourwe, “A Survey of Software Refactoring”, Journal IEEE Transactions on Software Engineering archive, vol.30, no.2, February 2004, pp.126-139.

[3] Yoshio Kataoka, Takeo Imai, Hiroki Andou, and Tet- suji Fukaya, “A Quantitative Evaluation of Maintainability Enhancement by Refactoring”, in Proceedings of the International Conference on Software Maintenance, Washington, DC, USA, 2002, pp. 576–585.

[4] J. A. Dallal and L. Briand, “A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes”, Journal ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 21, no. 2. March 2012.

[5] Tom Tourwe ́ and Tom Mens, “Identifying Refactoring Opportunities Using Logic Meta Programming”, European Conference on Software Maintenance and Reengineering, 2003.

[6] Liming Zhao, Jane Huffman Hayes “Predicating Classes in Need of Refactoring: An Application of Static Metrics”, Department of Computer Science, University of Kentuck, 2006.

[7] Raul Maticorna, Javier Perez, “Assisting Refactoring Tools Development Through Refactoring Characterization”, in Proceedings of the 6th International Conference on Software and Data Technologies, vol. 2, Seville, Spain, 18-21 July, 2011.

[8] Emerson Murphy-Hill, Andrew P. Black, Danny Dig, Chris Parnin, “Gathering Refactoring Data: a Comparison of Four Methods”, in Proceedings of the 2nd Workshop on Refactoring Tools, no.7, 2008.

[9] Jocelyn Simmonds, Tom Mens, “A Comparison of Software Refactoring Tools”, Programming Technology Lab, November 2002.

[10] S. Counsell Brunel Y. Hassoun G. Loizou R. Najjar, “Common Refactorings, a Dependency Graph and some Code Smells: An Empirical study of Java OSS”, in Proceedings of the ACM/IEEE international symposium on Empirical software engineering, 2006, pp.288 – 296.

[11] Don Roberts and John Brant, Ralph Johnson, “A Refactoring Tool for Smalltalk”, Theory and Practice of Object Systems - Special issue object-oriented software evolution and re-engineering archive, vol. 3, no. 4, 1997, pp. 253 – 263.
[12] Mincho Sandalski, Asya Stoyanova-Doycheva, Ivan Popchev, Stanimir Stoyanov, “Development of a Refactoring Learning Environment”, Cybernetics and Information Technology, vol.11, no 2, 2011.
[13] Deepak Advani, Youssef Hassoun & Steve Counsell, “Understanding the Complexity of Refactoring in Software Systems: a Tool-Based Approach”, International Journal of General Systems, vol.35, no.3, 2006,329-346.
[14] Shinpei Hayashi, Motoshi Saeki, Masahito Kurihara, “Supporting Refactoring Activities Using Histories of Program Modification”, IEICE - Transactions on Information and Systems archive, vol. E89-D no.4, April 2006, pp.1403-1412.
[15] Bryton, S.; Brito e Abreu, F.; Monteiro, M., “Reducing Subjectivity in Code Smells Detection: Experimenting with the Long Method”, Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference, Sept. 29 2010-Oct. 2 2010, pp.337- 342.
[16] Nidhi Tiwari. Using the Analytic Hierarchy Process (AHP) to Identify Performance Scenarios for Enterprise Application (2006).
[17] Saaty TL.The Analytic Hierarchy Process, McGraw-Hill, New York, (1980).
[18] Saaty, T.L. How to Make a Decision: the Analytic Hierarchy Process, Interfaces, Vol. 24, No. 6, pp.19--43 (1994).
[19] Karim O. Elish, Mohammad Alshayeb, “A Classification of Refactoring Methods Based on Software Quality Attributes”, Arabian Journal for Science and Engineering, vol. 36, no. 7, November 2011, pp. 1253-1267.
[20] Eduardo Kessler Piveta, Ana Moreira, Marcelo Soares Pimenta, João Araújo, Pedro Guerreiro, Roberto Tom Price, Ranking Refactoring Patterns Using the Analytical Hierarchy Process, in: proceeding of: ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, vol. ISAS-2, Barcelona, Spain, June 12-16, 2008.
[21] Robert K. Yin, “Case Study Research: Design and Methods: Applied Social Research Methods”, SAGE Publications, Inc; 2nd edition (March 18, 1994).
[22] Raimund Moser, Pekka Abrahamsson, Witold Pedrycz, Alberto Sillitti, Giancarlo Succi, “A Case Study on the Impact of Refactoring Quality and Productivity in a Agile Tam”, Balancing Agility and Formalism in Software Engineering Lecture Notes in Computer Science, vol.5082, 2008, pp 252-266.
[23] K. Stroggylos and D. Spinellis, “Refactoring - Does It Improve Software Quality?”, in Proceeding of the 5th International Workshop on Software Quality, 2007, p.10.
[24] Du Bois, B.; Demeyer, S.; Verelst, J., “Refactoring - Improving Coupling and Cohesion of Existing Code”, Reverse Engineering, in: Proceedings of the11th Working Conference, Nov 2004, pp.144-151.
[25] Raimund Moser1, Alberto Sillitti1, Pekka Abrahamsson2, and Giancarlo Succi1, “Does Refactoring Improve Reusability?” in Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components, 2006, pp.287-297.
[26] J.Ratzinger, M. Fischer, H. Gall, “Improving Evolvability Through Refactoring”, in Proceeding of the 2005 international workshop on mining software repositories, 2005, pp.1 – 5.
[27] E. Stroulia, R.V. Kapoor, “Metrics of Refactoring-based Development: An Experience Report”, In The seventh International Conference on Object-Oriented Information Systems, 2001, pp. 113–122.
[28] Y. Yu, J. Mylopoulos, E. Yu, J.C. Leite, L. Liu, E.H. D’Hollander, “Software Refactoring Guided by Multiple Soft- Goals”, in Proceedings of the 1st workshop on Refactoring: Achievements, Challenges, and Effects, in conjunction with the 10th WCRE conference 2003, Victoria, Canada, 2003, pp. 7-11.
[29] Kataoka, Y.; Imai, T.; Andou, H.; Fukaya, T., “A Quantitative Evaluation of Maintainability Enhancement by Refactoring”, Software Maintenance, in Proceedings of the International Conference, 2002, pp.576-585.
[30] Liming Zhao, Jane Huffman Hayes, “Rank-Based Refactoring Decision Support: Two Studies”, Innovations in Systems and Software Engineering, September 2011, vol.7, no. 3, pp 171-189.
[31] Geppert, B.; Mockus, A.; Robler, F., “Refactoring for Changeability: a Way to Go?”, Software Metrics, the 11th IEEE International Symposium, Sept. 2005, pp.10 -13.

[32] Robert K. Yin, “Qualitative Research from Start to Finish”, The Guilford Press; 1 edition (October 7, 2010).
[33] Rudiger, Lincke, “How do PhD Students Plan and Follow-up their Work? – A Case Study”, School of Mathematics and Systems Engineering, University of Sweden.
[34] Robert K. Yin, “Case Study Research: Design and Methods: Applied Social Research Methods”, SAGE Publications, Inc; 2nd edition (March 18, 1994).

Authors

Sultan Alshehri was born in Saudi Arabia in 1981; a PhD Student in Software Engineering Systems Department at University of Regina, Regina, Saskatchewan, Canada.

He worked as a computer science teacher in Riyadh, Saudi Arabia, 2004-2005. Currently, he is working as Programmer Analysts at SmarTech Company. In the same time, he holds the position of CEO for the LogicDots Company in Regina.

Mr. Alshehri holds a reward of a scholarship since 2005 until 2013 from the high Ministry of education in Saudi Arabia.

Dr. Luigi Benedicenti is the Associate Vice President (Academic) at the University of Regina. A full professor in the Faculty of Engineering at the University of Regina, he is a Professional Engineer licensed in Saskatchewan and a licensed Italian Engineer. His collaborative network extends beyond Saskatchewan with TRLabs and IEEE, and Canada through collaborative work with colleagues in Europe, South East Asia, and North America.

Benedicenti’s current research is in three areas: Software Agents, Software etrics, and New Media Technology. He envisions the unification of platform, tools, and optimizations for the provision of persistent distributed digital services, regardless of people’s location and delivery device.