International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 3 Issue 6 ǁ June 2015 ǁ PP.74-87

Ranking The Refactoring Techniques Based on The External Quality Attributes

Sultan Alshehri [1] and Abdulmajeed Aljuhani [2]
[1+2] *Department of Engineering, Software Engineering, University of Regina, Canada*

***ABSTRACT**: The selection of appropriate decisions is a significant issue that might lead to more satisfactory results. The difficulty comes when there are several alternatives and when all of them have the same chance of being selected. It is important, therefore, to find the kinds of priorities among all of these alternatives in order to choose the most appropriate one. The analytic hierarchy process (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. This paper presents an example of using the AHP to rank the refactoring techniques based on the external code quality attributes. XP encourages applying the refactoring where the code smells bad. However, refactoring may consume more time and efforts. Therefore, 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; Refactoring techniques; Analytic Hierarchy Process.*

I. Introduction:

Code refactoring is the process of re-designing existing code by changing its internal structure, and keeping its external behavior [1]. Refactoring is a core practice in extreme programming development process, which assists to enhance software design with reducing cost and effort of coding and testing [2].

1.1 Guidelines in the refactoring process

Several researchers, such as [3], exhibited various refactoring steps, which can be described as follows:

- Determining which part of the code that should be refactored.
- Selecting the appropriate refactoring methods to be applied.
- Applying the refactoring.
- Evaluating the influences of the applied refactoring methods on the code quality attributes [2].

Further steps can be found in [4,5,6].

Issue regarding refactoring tools:

Murphy-Hill et al. [7] conducted an empirical study, which leaded to collect data about refactoring process in order to assist in constructing a robust refactoring tool. Brunel et al. [8] analyzed several open-source java systems (JasperReports, Tyrant, Velocity, MegaMek, PDFBox, HSQLDB, Antlr) in order to investigate refactoring tools accuracy. Refactoring characterization has been introduced by Maticorna and Perez [9]; also, the authors explained how to use it as a comparing tool in different refactoring definitions. Roberts et al. [10] studied the functional requirements and empirical criteria for the refactoring tools, and their findings were that accuracy and the ability to search across the entire program are the most functional requirements, and integration and speed are the most empirical criteria. Simmonds and Mens [11] conducted a study to provide the strengths and weakness of four refactoring tools, which are SmalltalkWorks 7.0, Eclipse, Guru, and Together ControlCenter 6.0.

1.2 Identification of code smells to locate possible refactoring

Advani et al. [12] specified the field of complexity across the systems when applying refactoring. In addition, the authors introduced a method to assign the testing effort. Bryton et al. [13] presented Binary Logistic Regression (BLR) model in order to discover the code smell specifically Long Method objectively. Intelligent assistant and a refactoring agent has been used by Sandalski et al. [14] to inspect the refactoring architecture and evaluate the existing code in order to emphasize which part of the code needs to be refactored. Based on plug-ins for Eclipse, a technique has been introduced by Hayashi et al. [15] to assist the development team on how and where to do refactoring based on the histories of program modifications.

1.3 The impact of refactoring on the code quality attributes:

Section 5 in this paper, described several studies and 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.

II. The Analytical Hierarchy Process:

AHP is a multi-criteria decision making tool that structures a problem in a hierarchical model. Human factor plays a main role in AHP by gathering the elements of a problem and structuring them hieratically. Thomas Saaty has introduced AHP as a robust and sufficient method that can be used to solve complex decision-making problems [16]. AHP consists of several steps, such as:

- Breaking down the problem into hierarchal model.
- Identifying the criteria and designing criteria pairwise comparison matrix. Thus, the weight for each criterion will be specified with respect to the other criteria.
- Designing alternatives pairwise matrix with respect to the control criterion in each matrix.
- Calculating the consistency ratio in order to examine the consistency of the input errors.
- Selecting the highest weighted average rating after calculating the average weight for each alternative.

In order to design a pairwise comparison, Saaty [17] presented an importance scale to assign for each criterion and alternative. Table 1 shows the importance scale.

Table 1 AHP Numerical Scale Developed by Saaty [17].

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

III. Refactoring Techniques:

There are several benefits that can be achieved from the refactoring, such as increasing the programming speed, and detecting bugs [1]. Fowler [1] identified more than 70 refactoring techniques (i e. patterns), which grouped into six categories: moving features between objects, simplifying conditional expressions, dealing with generalization, making methods calls simpler, composing methods, and organizing data. For each refactoring pattern, there is a certain purpose and influence over the quality attributes. As mentioned above that using refactoring methods enhance the code and the design of the software; therefore, it is significant to determine the most important quality attributes in order to maximize the refactoring value. Since the lack of known how to use refactoring patterns, the procedure of choosing the appropriate refactoring patterns consumes more time and leads to conflict perspectives among team members. In this paper, we introduce AHP to rank various refactoring patterns regarding to their influences on the code quality. In addition, we selected eight refactoring patterns from four various categories introduced by Fowler n order to investigate the importance of the chosen refactoring patterns 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.

IV. Case Study Setup:

A part of this work has been published in [2]; therefore, the case study setup is similar to that in [2].

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 [18] 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 researchers 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.

V. AHP Use in Refactoring:

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

5.1 Background

In this section, we will highlight some of the previous related works that have studied the impact of the refactoring on the external code quality attributes as follow:
Leitch and Stroulia [19] studied the refactoring effects on software maintenance effort and costs using dependency analysis. Alshayeb [20] aimed to validate/invalidate the refactoring effects on some of the external quality attributes (adaptability, maintainability, understandability, reusability, and testability) in order to decide whether the cost and time put into refactoring are worthwhile. He concluded his study with results that stated that the refactoring does not necessarily improve these quality attributes.
Similarly, Raed and Li [21] used the hierarchal quality model to investigate the effect of refactoring activities on some software metrics such as reusability, flexibility, extendibility, and effectiveness. They found that not all the refactoring methods improve the quality factors.
Kataoka et al. [22] proposed a quantitative evaluation method to measure the maintainability enhancement by refactoring. They focused only on coupling metrics in order to quantify the refactoring effects.
Elish and Alshayeb [23,24] classified the refactoring techniques based on their effects on the internal and external quality attributes. The external quality metrics were adaptability, completeness, maintainability, understandability, reusability, and testability.
Weber and Reichert [25] proposed 11 refactoring techniques to support the business process management at the operational level and allow the designer to improve the quality of the process model. They focused specifically on the process-aware information system model (PAIS) that provides schemes for process execution.
Moser et al. [26] conducted a case study in industrial projects and agile environment to analyze the impact of refactoring on the reusability attribute. They analyzed how often part of the software (i.e. classes, methods, etc.) is used in a product.
Stroulia and Leitch [27] proposed a method to estimate the expected software maintenance cost by predicting the return on investment (ROI) for the refactoring activity. The authors claim that this would increase the adoption of refactoring practices and improve the quality attributes of the software, such as performance and maintainability. Finally, Moser et al. [28] developed a model to identify a refactoring effort during the maintenance phase.

5.2 Proposed Criteria for Ranking The Refactoring

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

- Reusability: defined as the capability for a component and subsystems to be suitable for use in more than one application, or in building other components, with little or no adaptation [29, 30].
- Flexibility: defined as " the ability of a system to adapt to varying environments and situations, and to cope with changes in business policies and rules. A flexible system is one that is easy to reconfigure or adapt in response to different user and system requirements" [31].
- Maintainability: defined as the ability of system to accept changes with a degree of ease. These changes could be modifying a component or other attribute to correct faults, improve performance, or adapt to a new environment [32].
- Understandability: defined as the degree to which the meaning of a software component is clear to a user [29].

5.3 AHP-refactoring structure for the external attributes

The top level is the main objective: ranking the refactoring techniques; the second level is the criteria:

reusability, flexibility, maintainability, and understandability; the third level is the alternatives: Extract Method, Inline Method, InlineTemp Method, Extract Class, Inline Class, Move Method, Pull Up Method, Rename Method. Figure 1 illustrates the AHP structure for the problem.

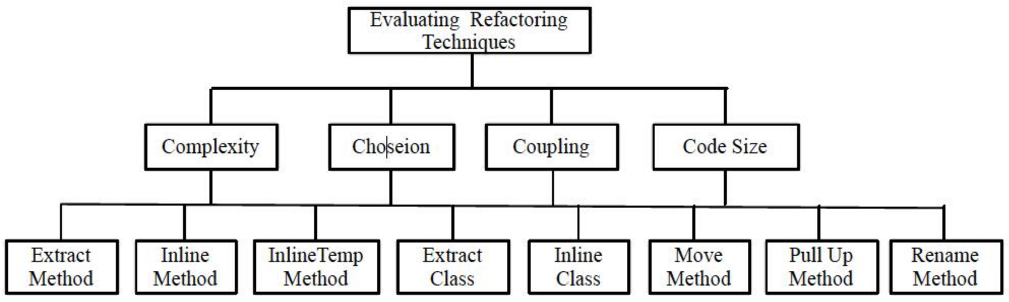

Figure 1 AHP Structure for the refactoring based on the external attributes.

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 first page was dedicated to collecting general information about the evaluator, his/her experience, and the type and the level of his/her programming skills. The participants compared the criteria using the Saaty scale from 1-9. The participants were asked as examples:

- Which is more important: reusability or flexibility and by how much?
- Which is more important: reusability or maintainability and by how much?
- Which is more important: reusability or flexibility and by how much?
- Which is more important: reusability or understandability and by how much?
- Which is more important: flexibility or maintainability and by how much?
- Which is more important: flexibility or understandability and by how much?
- Which is more important: maintainability or understandability and by how much?

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

- In term of flexibility, 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).

VI. AHP Evaluation Results

6.1 Educational Case Studies

For Team 1, the rankings for the refactoring techniques based on all criteria (i.e. reusability, flexibility, maintainability and understandability) are summarized as follows: First: Extract Class (18.33); Second: Rename Method (15.17); Third: Pull Up Method (14.13); Fourth: Extract Method (13.95); Fifth: Inline Class (12.81); Sixth: Move Method (12.25); Seventh: Inline Method (6.74); Eighth: Inline Temp Method (6.62). Table 2 summarizes the results. Figure 2 shows the importance of each criterion as follows: understandability (35.70), maintainability (30.09), reusability (22.42), and flexibility (11.80).

Table 2 Ranking the refactoring patterns by team 1.

Refactoring Techniques	All
Extract Class	18.33 %
Rename Method	15.17 %
Pull Up Method	14.13 %
Extract Method	13.95 %
Inline Class	12.81 %
Move Method	12.25 %
Inline Method	6.74 %
Inline Temp Method	6.62 %

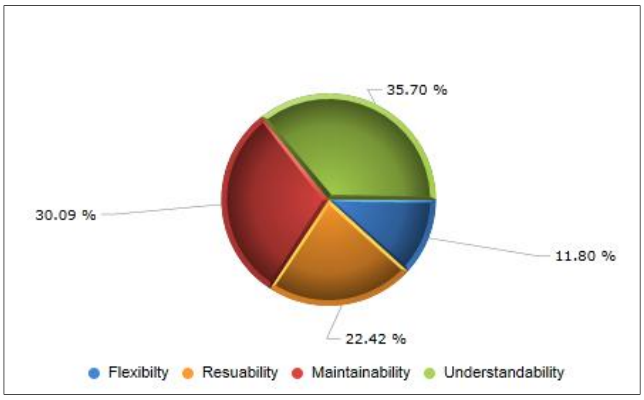

Figure 2 Importance of the external criteria for the Team 1.

The rankings for the prioritization of techniques by Team 2 is summarized as follows: First: Extract Class (16.66); Second: Extract Method (16.37); Third: Rename Method (15.41); Fourth: Pull Up Method (15.16); Fifth: Move Method (12.74); Sixth: Inline Temp Method (8.07); Seventh: Inline Method (7.92); Eighth: Inline Class (7.66). Table 3 summarizes the results.
Figure 3 shows the importance of each criterion as follows: maintainability (30.85), understandability (29.7), flexibility (21.77), and reusability (17.67).

Table 3 Ranking the refactoring patterns by team 2.

Refactoring Techniques	All
Extract Class	16.66 %
Extract Method	16.37 %
Rename Method	15.41 %
Pull Up Method	15.16 %
Move Method	12.74 %
Inline Temp Method	8.07 %
Inline Method	7.92 %
Inline Class	7.66 %

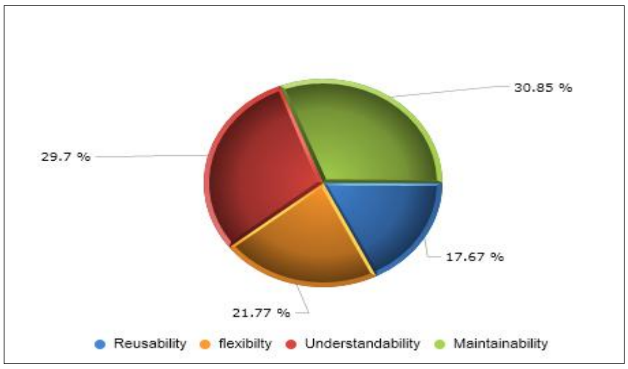

Figure 3 Importance of the external criteria by team 2.

6.1.1 Observations:

1. Considering all the criteria together, the Extract Class technique was ranked in the highest position by both teams, Team 1 and Team 2.
2. The Rename Method was ranked in advanced positions. Team 1 ranked it in the second position and Team 2 in the second position.
3. The understandability and maintainability quality attributes were considered the most important by both teams.
4. If we look at the refactoring techniques considering each criterion individually, we can see both teams ranked the Rename Method in the top position in the understandability attributes. For other criteria, each team has ranked the refactoring techniques differently. See tables 4 and 5.
5. For the reusability quality attribute, Team 1 ranked the Extract Class in the highest position, while Team 2 ranked the Extract Method at the top.
6. For the flexibility quality attribute, Team 1 ranked the Rename in the highest position, while Team 2 ranked the Extract Class at the top.
7. For the maintainability quality attribute, Team 1 ranked the Extract Class in the highest position, while Team 2 ranked the Pull Up Method at the top.
8. We can note the Extract Class was in the highest position for Team 1 in reusability and maintainability as individual criterion, while Team 2 considered it in the top only with the flexibility criterion.

Table 4 Refactoring Techniques based on each external criterion by team 1.

Refactoring Techniques	Resuability	Refactoring Techniques	Maintainability	Refactoring Techniques	Flexibility	Refactoring Techniques	Understandability
Extract Class	22.43 %	Extract Class	23.8 %	Rename Method	17.32 %	Rename Method	25.13 %
Extract Method	15.53 %	Extract Method	16.82 %	Extract Class	15.45 %	Extract Class	15.47 %
Pull Up Method	14.54 %	Pull Up Method	13.34 %	Pull Up Method	14.57 %	Inline Class	13.26 %
Move Method	13.89 %	Inline Class	10.87 %	Inline Class	11.99 %	Pull Up Method	11.91 %
Inline Class	11.46 %	Move Method	10.75 %	Extract Method	11.94 %	Move Method	11.75 %
Rename Method	9.72 %	Rename Method	8.32 %	Inline Temp Method	10.34. %	Extract Method	9.44 %
Inline Temp Method	6.35 %	Inline Temp Method	8.25 %	Inline Method	9.43 %	Inline Method	7.04 %
Inline Method	6.09 %	Inline Method	7.86 %	Move Method	8.95 %	Inline Temp Method	6.00 %

Table 5 Refactoring techniques based on each external criterion by team 2

Refactoring Techniques	Resuability	Refactoring Techniques	Maintainability	Refactoring Techniques	Flexibility	Refactoring Techniques	Understandability
Extract Method	22.43 %	Pull Up Method	21.75 %	Extract Class	20.74 %	Rename Method	22.82 %
Extract Class	15.53 %	Rename Method	17.18 %	Pull Up Method	14.92 %	Extract Method	17.39 %
Pull Up Method	14.54 %	Extract Method	14.4 %	Move Method	14.08 %	Move Method	16.72 %
Move Method	13.89 %	Extract Class	14.03 %	Extract Method	12.92 %	Extract Class	13.41 %
Rename Method	11.46 %	Inline Temp Method	8.61 %	Inline Method	10.52 %	Pull Up Method	8.4 %
Inline Temp Method	9.72 %	Move Method	8.6 %	Inline Class	10.38. %	Inline Temp Method	7.67 %
Inline Method	6.35 %	Inline Class	8 %	Inline Temp Method	8.27 %	Inline Method	7.27 %
Inline Class	6.09 %	Inline Method	7.43 %	Rename Method	8.18 %	Inline Class	8.31 %

6.2 Industrial case studies

The rankings for the prioritization of techniques by company A are summarized as follows: First: Inline Class (15.71); Second: Extract Method (15.05); Third: Extract Class (14.4); Fourth: Move Method (13.28); Fifth: Pull Up Method (12.78); Sixth: Inline Method (12.25); Seventh: Inline Temp Method (9.41); Eighth: Rename Method (7.12). Table 6 summarizes the results.

Figure 4 shows the importance of each criterion as follows: flexibility (35.97), maintainability (28.51), reusability (24.71), and understandability (10.81).

Table 6 Ranking the refactoring patterns by company A.

Refactoring Techniques	All
Inline Class	15.71 %
Extract Method	15.05 %
Extract Class	14.4 %
Move Method	13.28 %
Pull Up Method	12.78 %
Inline Method	12.25 %
Inline Temp Method	9.41 %
Rename Method	7.12 %

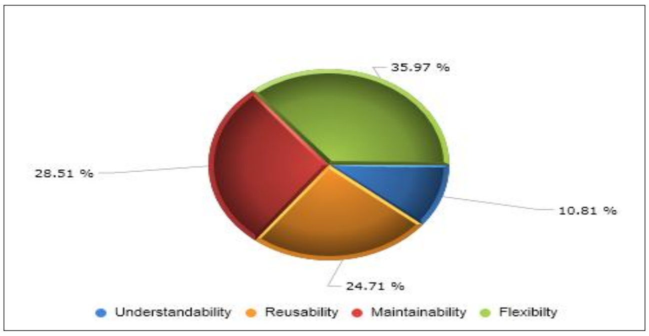

Figure 4 Importance of the external criteria for company A.

The rankings for the prioritization of techniques by company B are summarized as follows: First: Extract Class (27.84); Second: Extract Method (18.26); Third: Inline Class (11.07); Fourth: Move Method (10.37); Fifth: Inline Method (8.9); Sixth: Rename Method (8.81); Seventh: Pull Up Method (8.76); Eighth: Inline Temp Method (5.99). Table 7 summarizes the results. Figure 5 shows the importance of each criterion as follows: understandability (30.5), reusability (29.42), maintainability (26.97), and flexibility (13.11).

Table 7 Ranking the refactoring patterns by company B.

Refactoring Techniques	All
Extract Class	27.84 %
Extract Method	18.26 %
Inline Class	11.07 %
Move Method	10.37 %
Inline Method	8.9 %
Rename Method	8.81 %
Pull Up Method	8.76 %
Inline Temp Method	5.99 %

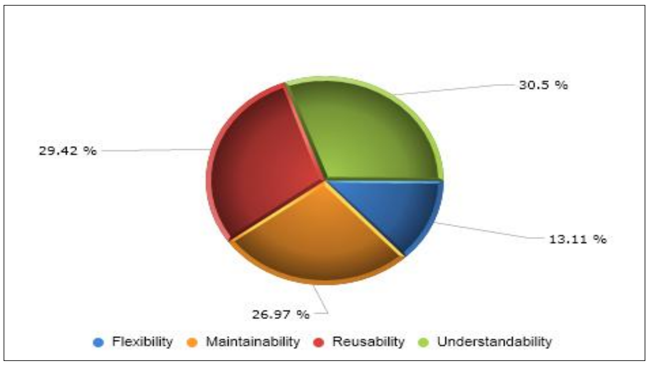

Figure 5 Importance of the external criteria for company B.

The rankings for the prioritization of techniques by company C are summarized as follows: First: Extract Class (21.7); Second: Inline Class (17.38); Third: Inline Method (11.83); Fourth: Extract Method (11.47); Fifth: Pull Up Method (10.86); Sixth: Move Method (9.7); Seventh: Inline Temp Method (9.35); Eighth: Rename Method (7.71). Table 8 summarizes the results. Figure 6 shows the importance of each criterion as follows: reusability (42.19), maintainability (26.34), understandability (18.76), and flexibility (12.71).

Table 8 Ranking the refactoring patterns by company C.

Refactoring Techniques	All
Extract Class	21.7 %
Inline Class	17.38 %
Inline Method	11.83 %
Extract Method	11.47 %
Pull Up Method	10.86 %
Move Method	9.7 %
Inline Temp Method	9.35 %
Rename Method	7.71 %

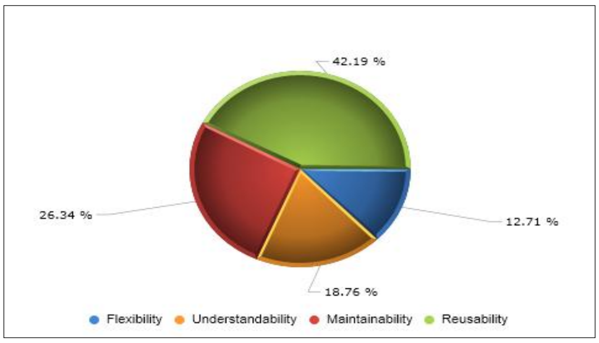

Figure 6 Importance of the external criteria for the refactoring by company C.

6.2.1 Observations:

1. Considering all the criteria together, the Extract Class technique was ranked in the highest position by companies B and C, while company A ranked the Inline Class in the highest position.
2. Extract Method was ranked in second position by companies A and B, while company C ranked the Inline Class in the second position.
3. The Rename Method was ranked in last positions by companies A and C, while company B ranked the Inline Temp Method in the last position. Also, A and C ranked the Inline Temp Method in the penultimate position.
4. Each company has ranked the quality attributes differently. Company A ranked maintainability and flexibility in the two top positions. Company B considered reusability and understandability to be the top ones. Company C considered reusability and maintainability the highest concerns. As result, companies A and C shared the same concerns about maintainability and ranked it high. On the other hand, companies B and C shared the concerns about reusability and ranked it to be one of the top criteria.
5. If we look at the refactoring techniques considering each criterion individually, we can see company B ranked the Extract Class in the top position in all the quality attributes: reusability, flexibility, maintainability, and understandability. Company C also ranked the Extract Class in the top position in terms of reusability and understandability, while company A ranked it in the top only in terms of maintainability. See tables 9, 10, and 11.
6. For the reusability quality attribute, Team 1 ranked the Extract Class in the highest position, while Team 2 ranked the Extract Method at the top.
7. Company C ranked the Inline Class in the first position in terms of flexibility and maintainability, while

company A ranked it at the top in reusability.

8. Company A ranked the Extract Method at the top in flexibility and the Move Method at the top in understandability.

Table 9 Refactoring techniques based on each external criterion by company A.

Refactoring Techniques	Resuability	Refactoring Techniques	Maintainability	Refactoring Techniques	Flexibility	Refactoring Techniques	Understandability
Inline Class	17.21 %	Extract Class	24.05 %	Extract Method	21.27 %	Move Method	15.71 %
Inline Method	16.62 %	Inline Class	19.87 %	Move Method	16.16 %	Pull Up Method	15.05 %
Pull Up Method	16.61 %	Inline Method	15.96 %	Pull Up Method	13.4 %	Extract Class	14.4 %
Extract Method	14.71 %	Inline Temp Method	13.97 %	Inline Class	13.08 %	Rename Method	13.28 %
Move Method	12.39 %	Extract Method	10.54 %	Extract Class	12.02 %	Inline Class	12.78 %
Extract Class	8.91 %	Move Method	7.47 %	Rename Method	8.67 %	Inline Temp Method	12.25 %
Rename Method	7.09 %	Pull Up Method	5.19 %	Inline Temp Method	8.49 %	Inline Method	9.41 %
Inline Temp Method	6.47 %	Rename Method	2.95 %	Inline Method	6.91 %	Extract Method	7.12 %

Table 10 Refactoring techniques based on each external criterion by company B.

Refactoring Techniques	Resuability	Refactoring Techniques	Maintainability	Refactoring Techniques	Flexibility	Refactoring Techniques	Understandability
Extract Class	27.83 %	Extract Class	31.17 %	Extract Class	19.86 %	Extract Class	30.21 %
Extract Method	17.08 %	Extract Method	20.93 %	Extract Method	15.87 %	Extract Method	18.77 %
Inline Class	13.8 %	Inline Class	10.81 %	Move Method	14.44 %	Rename Method	11.95 %
Move Method	9.74 %	Move Method	8.97 %	Inline Class	13.03 %	Pull Up Method	10.02 %
Inline Method	9.62 %	Rename Method	7.88 %	Inline Method	12.03 %	Move Method	9.58 %
Pull Up Method	8.62 %	Pull Up Method	7.84 %	Inline Temp Method	9.4. %	Inline Method	7.58 %
Rename Method	7.71 %	Inline Method	7.16 %	Pull Up Method	8.29 %	Inline Class	7.1 %
Inline Temp Method	5.61 %	Inline Temp Method	5.25 %	Rename Method	6.98 %	Inline Temp Method	4.8 %

Table 11 Refactoring techniques based on each external criterion by company C.

Refactoring Techniques	Resuability	Refactoring Techniques	Maintainability	Refactoring Techniques	Flexibility	Refactoring Techniques	Understandability
Extract Class	25.04 %	Inline Class	23.8 %	Inline Class	19.86 %	Extract Class	30.21 %
Extract Method	14.54 %	Extract Class	16.82 %	Extract Class	15.87 %	Inline Class	18.77 %
Move Method	12.53 %	Inline Method	13.34 %	Move Method	14.44 %	Inline Method	11.95 %
Pull Up Method	11.78 %	Pull Up Method	10.87 %	Inline Temp Method	13.03 %	Extract Method	10.02 %
Inline Class	10.34 %	Inline Temp Method	10.75 %	Inline Method	12.03 %	Pull Up Method	9.58 %
Inline Method	10.22 %	Rename Method	8.32 %	Rename Method	9.4. %	Inline Temp Method	7.58 %
Inline Temp Method	7.89 %	Extract Method	8.25 %	Extract Method	8.29 %	Rename Method	7.1 %
Rename Method	7.66 %	Move Method	7.86 %	Pull Up Method	6.98 %	Move Method	4.8 %

6.3 Impact of refactoring on the quality attributes

Tables 12 and 13 show the impact of the refactoring on the external quality attributes reported by Team 1 and Team 2.

Table 12 Refactoring impact on the external attributes by Team 1

	Reusability	Flexibility	Maintainability	Understandability
Extract Method	+	+	+	+
Inline Method	+	-	-	+
InlineTemp Method	0	0	0	0
Extract Class	0	0	0	0
Inline Class	-	-	+	0
Move Method	-	-	+	0
Pull Up Method	+	+	+	+
Rename Method	0	0	+	+

Table 13 Refactoring impact on the external attributes by Team 2

	Reusability	Flexibility	Maintainability	Understandability
Extract Method	+	+	-	+
Inline Method	-	-	+	+
InlineTemp Method	0	0	0	-
Extract Class	+	+	+	+
Inline Class	+	-	+	+
Move Method	-	0	+	+
Pull Up Method	0	0	+	+
Rename Method	+	0	+	+

VII. REFACTORING VALIDATION

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 and two industrial studies to validate the AHP evaluation results obtained previously. The collected information included the following:

• The impact of the proposed refactoring techniques on the 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.

7.1 Number of times using the refactoring techniques

Tables 14 and 15 summarize how many times each of the refactoring techniques were used in each iteration by companies A and B.

Table 14 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 15 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 external quality attributes

Table 16 and 17 summarize the external impacts for each refactoring technique for companies A and B.

Table 16 Refactoring impact on the external attributes by company A

	Reusability	Flexibility	Maintainability	Understandability
Extract Method	+	+	-	+
Inline Method	0	-	-	-
InlineTemp Method	-	-	-	-
Extract Class	+	+	+	+
Inline Class	-	+	+	-
Move Method	+	+	+	+
Pull Up Method	0	0	+	+
Rename Method	NA	+	+	+

Table 17 Refactoring impact on the external attributes by company B

	Reusability	Flexibility	Maintainability	Understandability
Extract Method	+	+	-	+
Inline Method	0	-	-	+
InlineTemp Method	-	-	-	+
Extract Class	+	+	+	+
Inline Class	-	-	-	0
Move Method	-	-	-	0
Pull Up Method	+	+	+	+
Rename Method	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 18 shows that the time was reduced.

Table 18 Refactoring impact on time for company A and B

	Time before AHP	Time 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 external quality attributes. Tables 15 and 16 show that Extract Method and Extract Class are commonly used and have positive effects (as seen 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 Method had a positive impact on all the attributes in both companies' results. The reusability, flexibility, and understandability increased; the maintainability effort was reduced. For both companies, Extract Class increased reusability, flexibility, and understandability, which is a good result. However, the maintainability also increased, which is a negative impact.
- 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 external 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.

VIII. 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. 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. 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.

IX. 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 studies were regarding the time efficiency (59%, 62%, 61%, 57%, 57%). See table 19.

Table 19 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; Team 1 (74%), Team 2 (93%), company A (93%), company B (93%), and company C (94%).
- Second: creating an informative discussion and learning opportunities, Team 1 (71%), Team 2 (88%), company A (86%), company B (95%), and company C (93%).
- Third: clarifying the ranking problem; Team 1 (71%), Team 2 (90%), company A (86%), company B (93%), and company C (83%).
- Fourth: resolving the conflicting opinions among members; Team 1 (64%), Team 2 (86%), company A (86%), company B (93%), and company C (83%).
- Fifth: elevating the team performance; Team 1 (76%) and Team 2 (88%), company A (79%), company B (86%), and company C (not study).

X. VALIDITY

Construct validity, internal validity, external validity and reliability describe common threats to the validity tony performed study [33].”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” [34]. 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 [35]. 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 [35]. 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.

XI. 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 reusability, flexibility, maintainability and understandability 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 external 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] Alshehr, Sultan, Luigi Benedicenti, “Ranking the Refactoring Techniques Based On The Internal Quality Attributes”.

[3] 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.

[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] 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.

[8] 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.

[9] 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.
[10] 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.
[11] Jocelyn Simmonds, Tom Mens, "A Comparison of Software Refactoring Tools", Programming Technology Lab, November 2002.
[12] 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.
[13] 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.
[14] Mincho Sandalski, Asya Stoyanova-Doycheva, Ivan Popchev, Stanimir Stoyanov, "Development of a Refactoring Learning Environment", Cybernetics and Information Technology, vol.11, no 2, 2011.
[15] 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.
[16] Saaty TL.The Analytic Hierarchy Process, McGraw-Hill, New York, (1980).
[17] Saaty, T.L. How to Make a Decision: the Analytic Hierarchy Process, Interfaces, Vol. 24, No. 6, pp.19--43 (1994).
[18] Robert K. Yin, "Case Study Research: Design and Methods: Applied Social Research Methods", SAGE Publications, Inc; 2nd edition (March 18, 1994).
[19] Leitch, R.; Stroulia, E., "Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis", Software Metrics Symposium, in Proceedings of the Ninth International, 3-5 Sept.2003, pp.309-322.
[20]. M. Alshayeb, "Empirical Investigation of Refactoring Effect on Software Quality", Information and Software Technology, vol. 51, no.9, September 2009, pp.1319–1326.
[21] Raed Shatnawi, Wei Li, " An Empirical Assessment of Refactoring Impact on Software Quality Using Hierarchical Quality Model", International Journal of Software Engineering and Its Applications, vol. 5, no.4, October 2011.
[22] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring", in Proceeding of the International Conference Software Maintenance, October, 2002, pp. 576-585.
[23] Karim O. Elish and Mohammad Alshayeb, "Using Software Quality Attributes to Classify Refactoring to Patterns", Journal of Software, vol. 7, no. 2, February 2012.
[24] Alshayeb, M., H. Al-Jamimi and M. Elish, "Empirical Taxonomy of Refactoring Methods for Aspect-Oriented Programming", Journal of Software Maintenance and Evolution: Research and Practice, incorporating Software Process: Improvement and Practice, accepted March 2011.
[25] Weber, B. and Reichert, M, "Keeping the Cost of Process Change Low through Refactoring", Technical Report TR-CTIT-07-86, Centre for Telematics and Information Technology University of Twente, Enschede, 2007.
[26] Raimund Moser, Alberto Sillitti, Pekka Abrahamsson, Giancarlo Succi "Does Refactoring Improve Reusability?", in Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components, 2006, pp. 287-297.
[27] E Stroulia and Leitch, "Understanding the Economics of Refactoring", in Proceedings of the Fifth ICSE Workshop on Economics-Driven Software Engineering Research, 2003.
[28] Raimund Moser, Witold Pedrycz, Alberto Sillitti,Giancarlo Succi, "A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories", in Proceedings of the 9th international conference on Product-Focused Software Process Improvement, 2008, pp. 360-370.
[29] W. Salamon and D. Wallace, "Quality Characteristics and Metrics for Reusable Software (preliminary Report)", US DoC for US DaD Ballistic Missile Defense Organization, NISTIR 5459, May 1994.
[30] Mohammad Alshayeb, "The Impact of Refactoring to Patterns on Software Quality Attributes", The Arabian Journal for Science and Engineering, June 2010.
[31] J.D. Meier, Alex Homer, David Hill, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher, "Patterns & Practices Application Architecture Guide 2.0", 2008.
[32] IEEEStd.610.12, Std. 610.12 - IEEE Standard Glossary of Software Engineering Terminology: The Institute of Electrical and Electronics Engineers, 1991.
[33] Robert K. Yin, "Qualitative Research from Start to Finish", The Guilford Press; 1 edition (October 7, 2010).
[34] Rudiger, Lincke, "How do PhD Students Plan and Follow-up their Work? – A Case Study", School of Mathematics and Systems Engineering, University of Sweden.
[35] 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.

Abdulmajeed Aljuhani is a PhD student in Software Systems Engineering Department at University of Regina, Regina, Saskatchewan, Canada.
Currently, he is working as a lecturer at Tibah University, Medina, Saudi Arabia. Mr. Aljuhani holds a reward of a scholarship since 2009 until 2017 from the high Ministry of education in Saudi Arabia.