A Multiple-Criteria Decision Making Model for Ranking Refactoring Patterns

Abdulmajeed Aljuhani
Faculty of Engineering and Applied Science
University of Regina,
Regina, Canada

Luigi Benedicenti
Faculty of Computer Science
University of New Brunswick,
Fredericton, Canada

Sultan Alshehri
Computer Science and Information Technology College
Majmaah University
Majmaah, Saudi Arabia

***Abstract*—The analytic network process (ANP) is capable of structuring decision problems and finding mathematically determined judgments built on knowledge and experience. Researches suggest that ANP can be useful in software development, where complicated decisions happen routinely. In extreme programming (XP), the refactoring is applied where the code smells bad. This might cost more effort and time. As a result, in order to increase the advantages of refactoring in less effort and time, the analytic network process has been used to accomplish this purpose. This paper presents an example of applying the ANP in order to rank the refactoring patterns regarding the internal code quality attributes. A case study that was conducted in an academic environment is presented in this paper. The results of the case study show the benefits of using the ANP in XP development cycle.**

***Keywords*—Analytic network process; extreme programming; refactoring practice; refactoring patterns**

I. Introduction

The process of enhancing the structure of an existing code by altering the internal design without changing the external design is called code refactoring [1]. It is a significant issue in the XP development cycle to enhance software design, and to minimize the cost and effort that are needed for testing and coding. Some researchers have concentrated on guidelines for the refactoring process; for example, a 3-stage model has been introduced by Kataoka *et al.* [2], and the model contains "identification of refactoring candidates, validation of refactoring effects, and application of refactoring" [2]. Meanwhile, Mens and Tourwe [3] have described the refactoring phases in more detail. These phases start with specifying the portion of the software that should be refactored, then determining which refactoring technique is suitable to be applied, performing the refactoring, and finally measuring the influence of the applied refactoring technique on the code quality [3]. Other researchers investigated various aspects of refactoring techniques. Simmonds and Mens [4] studied four software-refactoring methods: Eclipse, Together ControlCenter 6.0, SmalltalkWorks 7.0, and Gurn. In addition, Murphy-Hill *et al.* [5] conducted an empirical investigation to compare four techniques. These four techniques were applied to collect refactoring data in order to assist in establishing a powerful refactoring technique.

Maticorna and Perez [6] introduced refactoring interpretation and the possibility of using it as a method in order to compare various refactoring explanations, involving refactoring catalogs. Moreover, Maticorna and Perez [6] have worked on various refactoring concerns, like actions, application on scheduling, design, and scope, which might lead to the building of refactoring tools.

Several open-source Java systems have been investigated by Brunel et al. [7] in order to measure the accuracy of refactoring methods. This measuring is done by examining the following Java systems: MegaMek, Velocity, Antlr, HSQLDB, PDFBox, Tyrant, and JasperReports. Murphy-Hill [8] built a model to investigate how refactoring techniques work in terms of the style of the refactoring browser. The model is made up of the following phases: identify, initiate, and execute [8].

Roberts *et al.* [9] examined the practical factors and technical requirements for the refactoring techniques. The authors emphasized that the ability to search the whole program and the accuracy are the most technical requirements. In addition, integration and speed are the most practical factors.

Marija and Kresimir [10] evaluated seven refactoring tools in order to choose the most suitable one. The seven tools were: Refactoring Browser (Smalltalk), Eclipse (C++, Java), Refactor (C# VB.NET, C++, ASP.NET), IntelliJ Idea (Java), Refactor (C# VB.NET, ASP.NET), NDepend (.NET code base), and Refactor (C++, Java). These refactoring tools were compared to each other concerning various issues, such as reliability, scalability, automation, discover-ability, coverage, and configurability.

Mahmood and Reddy [11] examined three refactoring techniques in order to avoid human errors while performing the manual refactoring. The authors evaluated the following refactoring tools: JBuilder 2008, RefactorIT 2.7 beta, and IntelliJ Idea 7.0.4. The authors compared these techniques with respect to various issues, such as user control, consistency, information processing, user experience, goal assessment, errors, design for the user, and ease of use. The authors proposed some enhancements in order to maximize the consistency of software usability.

Other studies focused on the identification of code smells in order to locate possible refactoring. For example, Hayashi et al. [12] introduced a tool for Eclipse using plug-ins. This tool directs the developer in terms of how to perform refactoring and which part of the code uses the histories of program

modification. The proposed tool focused on answering the following questions: Where to refactor? Which suitable refactoring technique should be used? When should refactoring be applied?

II. The ANP

The Analytic Network Process (ANP) is a multi-criteria approach of estimation used to infer relative need sizes of supreme numbers from singular judgments (or from genuine estimations standardized to a relative frame) that likewise have a place with a central size of outright numbers [13]. The ANP gives a structure to show an solution for a specific problem, which prompts a choice for that issue. In the ANP technique, dependencies among different criteria are considered making it not the same as the Analytic Hierarchy Process (AHP) [13]. Saaty stated that in truth the ANP utilizes a system without the need to indicate levels. As in, the AHP, strength or the relative significance of impact is a focal idea [13]. In the ANP, one structures a judgment from the principal size of the AHP by noting two sorts of inquiries with respect to quality of strength: 1) Given a rule, which of two components is more overwhelming concerning that basis? 2) Which of two components impacts a third component more, as for a measure [13]?

In pairwise comparisons, entered values reflect the relative impact among components regarding a control paradigm. These entered values depend on the significance of every criterion. As such, the ANP is a helpful approach for forecast and for representing to an assortment of contenders with their expressly known and verifiably accepted cooperations and the relative qualities with which they use their impact in making a decision. It is likewise helpful in struggle determination where there can be many contradicting impacts [13]. The system structure comprises of various clusters, and these clusters contain different nodes or components. These clusters are associated with each other in view of the relative impacts among the nodes. The connections can either have outer relative impact, which implies components in cluster X influence component in cluster Y, or interior relative impact, which implies components in a similar cluster (e.g.X) influence each other. For this situation, the outside relative impact is named external reliance, and the interior relative impact is named internal reliance [13]. The network structure permits criticism models through cycle association, and the ANP gives distinctive sorts of nodes, for example, source, middle, and sink. Again, as indicated by Saaty that a source node is a starting point of ways of impact (significance) and never a goal of such ways. A sink node is a goal of ways of impact and never a root of such ways. A full network can incorporate source nodes; middle of the road nodes that fall on ways from source nodes, lie on cycles, or fall on ways to sink nodes; lastly sink nodes [14]. Fig. 1 gives a general idea of the ANP structure [14].

Another part of the ANP structure is the organizing of various alternatives keeping in mind the end goal to make a suitable decision. This begins by making pairwise comparisons, in light of a principal scale, as appeared in Table I. Following this, the vector of priorities is the foremost eigenvector of the matrix. This vector gives the relative priority of the criteria measured on a ratio scale. That is, these priorities

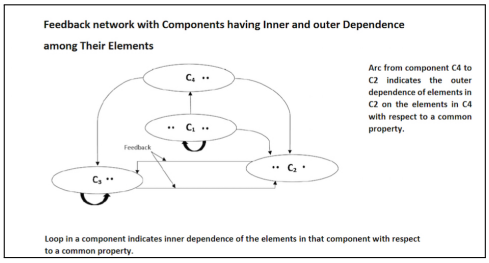

Fig. 1. The analytic network process structure [14].

TABLE I. ANP Fundamental Scale Developed by Saaty [15]

Scale	Numerical rating	Reciprocal
Equal importance	1	1
Moderate importance of one over other	3	$\frac{1}{3}$
Very strong or demonstrated importance	7	$\frac{1}{7}$
Extreme importance	9	$\frac{1}{9}$
Intermediate values	2,4,6,8	$\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}$

TABLE II. Random Index [14]

Order	1	2	3	4	5	6	7	8	9	10
R.I	0	0	0.52	0.89	1.11	1.25	1.35	1.4	1.45	1.49

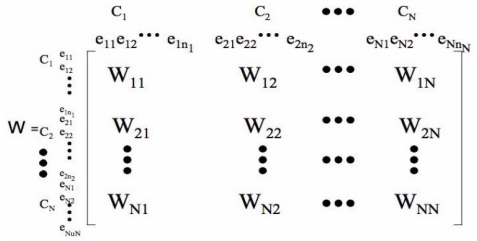

Fig. 2. The Super-matrix of a network [14].

are remarkable inside augmentation by a positive consistent. In the event that one guarantees that they whole to one they are then extraordinary and have a place with a size of supreme numbers [14]. "The consistency index of a matrix is given by C.I. (max n)/(n-1), where n is the number of alternatives. The consistency ratio (C.R.) is obtained by forming the ratio of C.I. The suitable group of numbers is exihibited in Table II, each of which is an average random consistency index computed for n 10 for very large samples. They create randomly generated reciprocal matrices using the scale $\frac{1}{9}$, $\frac{1}{8}$, $\frac{1}{2}$, 1, 2, 8, 9 and calculate the average of their eigenvalues. This average is used to form the Random Consistency Index R .I" [14]. The consistency proportion (C.R) ought to be lower than 0.10 (or 0.20), something else, the entered judgements should be improved.

In the wake of getting all priorities from the pairwise comparisons, these priorities are set in a supermatrix. As per Saaty [14] the supermatrix represents the impact priority of a component on the left of the matrix on a component at the top of the matrix as for a specific control rule. A supermatrix alongside a case of one of its general passage matrices is appeared in Fig. 2. The segment C1 in the supermatrix incorporates all priority vectors inferred for nodes that are parent nodes in the C1 cluster [14].

III. Refactoring Techniques

Refactoring assists the development team to enhance the software design, understand the software more easily, find errors, and program faster, as Fowler *et al.* [1] confirmed. Fowler *et al.* [1] specified several refactoring techniques and arranged them into the following categories: moving features between objects, making method calls simpler, simplifying conditional expressions, composing methods, dealing with generalization, and organizing data. Each of these categories influences the quality attributes. Each project might have different quality attribute priorities, and using the refactoring techniques enhances the software design and the code. Therefore, in order to maximize the benefit from the system, it is important to assign the developers' efforts to the most significant quality attributes. Selecting the refactoring techniques consumes time and might lead to conflicting opinions.

In this paper, the main objective is to rank refactoring techniques according to their influence on the internal code quality attributes. Five refactoring techniques have been selected in this study, in order to examine their importance using the ANP. These techniques were selected from the four different groups introduced by Fowler *et al.* [1]. The selected refactoring techniques are: Extract Method, Extract Class, Inline Class, Pull UP Method, and Rename Method.

IV. Methodology

The main objective in this research is to investigate how the analytic network process might be used to rank the refactoring patterns in order to determine the most suitable one for the software project. The case study methodology, which is explained in [16], is the research methodology.

The following research questions provide more focus for the research case study:

1) What is the significance of engaging the ANP when applying refactoring?
2) How can refactoring patterns be ranked using the ANP?
3) How does the ANP influence the development team's communication and productivity in the refactoring practice?
4) How can the development team reduce time when refactoring using the ANP?

Moreover, the study propositions are as follows:

Proposition 1: *The ANP catches significant criteria and alternatives that affect refactoring patterns.*

Proposition 2: *The ANP supports ranking and selection activities in the refactoring practice.*

Proposition 3: *The ANP includes creative debate and enhances team communication.*

Proposition 4: *The ANP focuses on the most valuable refactoring methods in order to increase the quality of the code.*

Proposition 5: *The ANP clarifies conflicting perspectives between the development teams when performing refactoring.*

From the above inquiries, we determined the units of analysis for our investigation. The primary target is ranking different XP refactoring patterns in regards to the inside quality attributes. Properly, assessing and ranking are two units of analysis. Another is the members' point of view of the ANP benefits in refactoring practice. Hence, the plan of this case study incorporates numerous cases, installed with different units of analysis. The rationale connecting of the gathered information to the study propositions is appeared at the end of this paper.

V. Data Collection and Sources

At the beginning of each use for the ANP in extreme programming, we investigate the ANP benefits and ability by introducing the related criteria and extreme programming areas. Data is gathered from looking past studies and literature review. Too, data triangulation is gained with a specific end goal to expand the validity of the study.

The main data origin of this paper is an extreme programming project, developed during the winter semester of 2016 at the University of Regina. The data sources in this research are:

- Questionnaires given to the students during the development of the XP project.
- Archival records, such as study plans, from the students.
- Comments from the customer.
- Open-ended interviews with the students.

VI. Case Study

At the beginning, the authors would like to address that a part of this case study has been published in [17]. The case study was organized during a 12-week Winter 2016 semester at the University of Regina. Several researches, as [18], [19] and [20], tended to that the reasonable XP team size is in the vicinity of three and seven individuals. In addition, Ambler [21] accentuated that the accomplishment of agile project is 83 % with group estimate under eleven individuals, and the rate runs bring down with expanding the group measure for more than eleven individuals [21]. The significant reason for this diminishing in the achievement rate is in regards to correspondence need or misconstruing with the the large team size. In this way, we had 12 graduate students from the University of Regina, and one extra member, a customer, who was incorporated into this case study. These students had transitional information of XP process and practices, and various programming levels. The dominant part of these students was a part of an expert program, implying that their graduate degree was a part of their expert development and that they had past work involvement in the software industry. Some of these students were proceeding to work part-time. The members' experiences included different programming languages, for example, C++, Java, and PHP. The members were sorted out into two groups, the principal group utilized the ANP strategy with a specific end goal to make their decisions in ranking the refactoring techniques, and the second group took after the traditional XP way, which is based on voting, for their decisions. The two groups were made a request to develop an

XP project called"Professors' Availability Managing System" finish with an arrangement of requirements. The undertaking was produced in 5 iterations, permitting two weeks for each. Toward the finish of the project, the two groups actualized all framework requirements. The members were asked to working on refactoring techniques amid the improvement cycle to rank them. Help materials that concentrated on refactoring practice were given to the members so as to guarantee their comprehension. The ANP group was given white papers, a few introductions, and other imperative materials about the ANP keeping in mind the end goal to enable them to apply it in their development. Team 1 honed on a few pairwise comparisons and expanded their understandings of the ANP structure. At the end, the researcher handed out a questionnaire to the participants in order to gather more data about the members' points of view.

VII. Applying ANP in Refactoring

The main objective of applying the ANP in refactoring is to assist the XP team members in ranking the refactoring techniques with respect to the code quality attributes. In this paper, the ANP is used to rank the refactoring techniques based on internal quality attributes. The following sections present the ANP structure, evaluation and process.

A. Background

This section will introduce some previous studies that have examined the effect of refactoring techniques on the internal code quality attributes. This is following by introducing the ANP applying to rank the refactoring techniques.

Zhao and Hayes [22] conducted two case studies in order to investigate an approach that specifies which packages and classes need to be refactored according to various measures, like complexity, coupling, and code size. Using a measure-driven refactoring decision, the authors presented a rank-based software in order to support the team members' decisions about where resources can be applied during refactoring.

Dallal and Briand [23] presented an automated refactoring method to enhance the cohesion of the software in order to enhance program testability. Sahraoui et al. [24] organized an empirical study in order to examine the effect of coupling and inheritance metrics on maintainability. The authors discovered a portion of the system that needed to be enhanced and refactored.

Stroulia and Kapoor [25] studied the possibility of enhancing the design and code quality using refactoring. Several refactoring techniques, such as Extract Abstract Class and Extract Superclass, were applied, and the results showed decreases in the number of methods, the number of statements, lines of code, and the number of collaborators in the individual system classes.

Moser et al. [26] organized a case study to investigate the influence of refactoring on the internal quality attributes of source code. The case study was done in an Agile environment, and the selected quality attributes were coupling, response of class, number of children, number of methods per class, depth of inheritance tree, cohesion, and complexity. Based on their proposed method, the authors found that refactoring might enhance the internal metrics of object-oriented classes that are written in Java for reusability.

Bois and Mens [27] introduced a framework for the internal code qualities, like cohesion, number of children, number of methods, coupling, and response for a class. In order to achieve this, the authors investigated various refactoring techniques, such as Encapsulate Filled, Pull Up Method, and Extract Method.

Elish and Alshayeb [28] categorized refactoring patterns according to their influence on external and internal code quality attributes. The authors selected the following refactoring techniques: Form Template Method, Replace Construction with Creation Methods, Replace Conditional Dispatcher with Command, Chain Constructors, Introduce Null Object, Unify Interface, and Compose Method. The authors investigated different internal code quality metrics such as Number of Test Cases (NOTC) for the size of test case, Lines of Code for Class (LOCC), FOUT, LOC, DIT, LCOM, Number of Methods (NOM), Number of Fields (NOF), Number of Children (NOC), RFC, and WMC.

Over the course of 15 months, Ratzinger et al. [29] evaluated an industrial system. The authors exhibited the way that refactoring could improve the software evolvability and minimize the change couplings. In addition, Kataoka et al. [30] emphasized that refactoring patterns like extract class and extract method enhance system maintainability and minimize coupling in the code.

B. Proposed Criteria for Ranking the Refactoring Techniques

It is important to specify code quality attributes in order to rank refactoring techniques. The code quality attributes should be identified based on their value to the organization or the team member. Different projects will have different factors and alternative refactoring techniques to be examined. In this thesis, there are four internal code quality attributes that are selected as criteria used to rank the refactoring patterns:

- **Complexity:** The degree of connectivity among components of a design unit [31].
- **Cohesion:** Each component implements one function and implements it well [31].
- **Code Size:** Size in terms of number of files, number of lines of code (#LOC), functions, tables, classes, etc. [31].
- **Coupling:** The strength of the interconnections between the system components [31].

C. ANP Structure for Ranking Refactoring Methods Based on the Internal Attributes

Structuring the problem as a network that consists of three clusters is the first step in the ANP. The first cluster is the objective, which ranks the refactoring patterns. The second cluster contains the criteria: coupling, code size, complexity, and cohesion. The third cluster includes the alternatives: Pull Up Method, Extract Class, Rename Method, Extract Method, and Inline Class. Fig. 3 shows the ANP structure for the problem.

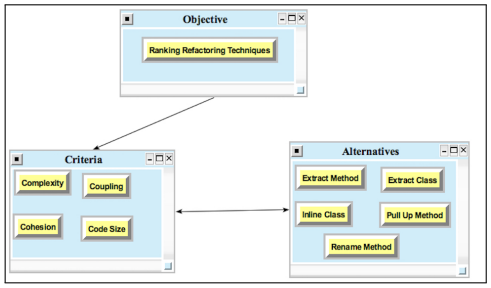

Fig. 3. ANP network for ranking refactoring techniques based on the internal attributes.

TABLE III. RANKING THE REFACTORING PATTERNS BASED ON INTERNAL ATTRIBUTES BY TEAM 1

Refactoring Patterns	Scores (%)
Extract Method	29.17 %
Extract Class	25.01 %
Pull Up Method	18.74 %
Inline Class	17.58 %
Rename Method	9.48 %

D. Pairwise Comparisons for the Refactoring Techniques

The participants have applied the refactoring techniques in their XP project in order to note the effect on the code. After that, based on the proposed criteria, the students evaluated each refactoring technique. The ANP team received the suitable ANP papers and tables in order to facilitate the comparisons process. Examples of the participants' questions are:

- With respect to Extract Method: which criterion is more important, cohesion or coupling and by how much?
- With respect to Extract Class: which criterion is more important, complexity or cohesion and by how much?
- With respect to cohesion: which method do you prefer, Extract Method or Extract Class?
- With respect to coupling: which method do you prefer, Extract Method or Extract Class?

The same comparisons and questions were done again for all refactoring techniques and code quality attributes.

VIII. FINDING AND RESULTS

Team 1's results of ranking the refactoring patterns with respect to all four criteria is as follows: first, Extract Method; second, Extract Class; third, Pull Up Method; fourth, Inline Class; and fifth, Rename Method. Table III shows the scores of each pattern. Team 1 ranked cohesion as the most important criterion, followed by complexity in the second position, while code size and coupling were ranked in the third and fourth positions, respectively. Fig. 4 exhibits the importance of each criterion as a percentage according to Team 1.

Team 2 ranked the refactoring patterns as follows: first, Extract Class; second, Extract Method; third, Inline Class; fourth, Pull Up Method; and fifth, Rename Method. Table IV displays the ranking of refactoring patterns by Team 2. Moreover, in terms of the most important criterion, Team 2 ranked coupling in the first position. Table V shows the ranking of the criteria by Team 2.

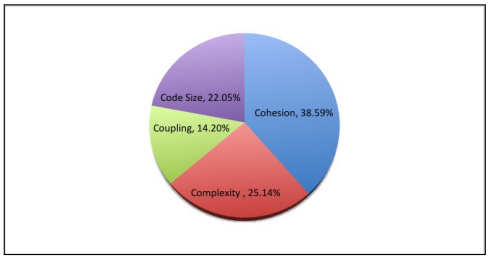

Fig. 4. The importance of the internal attributes for the refactoring patterns by Team 1.

TABLE IV. REFACTORING TECHNIQUES RANKING BY TEAM 2

Ranking	Refactoring Techniques
1	Extract Class
2	Extract Method
3	Inline Class
4	Pull Up Method
5	Rename Method

TABLE V. THE IMPORTANCE OF THE CRITERIA BY TEAM 2

Ranking	Criteria
1	Coupling
2	Cohesion
3	Complexity
4	Code Size

A. Observations

1) With respect to all of the criteria, Team 1 ranked the Extract Method as the highest refactoring technique.
2) Team 2 ranked Extract Class as the highest refactoring technique.
3) Both teams ranked Rename Method in the last position.
4) Team 1 ranked cohesion as the most important criterion, while Team 2 ranked coupling as the most important criterion.
5) With respect to each criterion individually, Team 1 ranked the Pull Up Method highest in terms of code size. Table VI shows the ranking of all refactoring techniques with respect to each criterion.
6) Inline Class was ranked highest with respect to the coupling criterion by Team 1.
7) With respect to each refactoring technique individually, we can see that reducing complexity was ranked highest according to Rename Method. Table VII shows the weight of each criterion with respect to each refactoring pattern.

IX. REFACTORING VALIDATION

Based on the previous ANP evaluation, we discovered that three refactoring patterns received high rankings: Extract Method, Extract Class, and Rename Method. This section shows several collected technical information in order to validate the ANP evaluation findings.

TABLE VI. REFACTORING TECHNIQUES WITH RESPECT TO EACH CRITERION FOR TEAM 1

Techniques	*Cohesion*	*Techniques*	*Coupling*	*Techniques*	*Code-Size*	*Techniques*	*Complexity*
Extract Method	53.09 %	Inline Class	46.23 %	Pull Up Method	41.22 %	Extract Class	47.30 %
Extract Class	28.42 %	Pull Up Method	28.51 %	Inline Class	31.20 %	Extract Method	23.16 %
Pull Up Method	9.26 %	Rename Method	11.96 %	Rename Method	13.44 %	Inline Class	11.38 %
Rename Method	5.91 %	Extract Method	7.75 %	Extract Method	7.96 %	Rename Method	10.10 %
Inline Class	3.29 %	Extract Class	5.52 %	Extract Class	6.16 %	Pull Up Method	8.04 %

TABLE VII. CRITERIA WEIGHTS WITH RESPECT TO EACH REFACTORING TECHNIQUE FOR TEAM 1

	Extract Method	Extract Class	Inline Class	Pull Up Method	Rename Method
Cohesion	64.32 %	66.36 %	7.01 %	5.48 %	15.54 %
Coupling	11.32 %	10.36 %	14.13 %	26.61 %	6.29 %
Code-Size	20.28 %	18.24 %	22.33 %	34.88 %	6.29 %
Complexity	4.06 %	5.02 %	56.51 %	33.01 %	71.86 %

TABLE VIII. REFACTORING PATTERNS EFFECT ON THE INTERNAL ATTRIBUTES BY TEAM 1

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

TABLE IX. REFACTORING PATTERNS EFFECT ON THE INTERNAL ATTRIBUTES BY TEAM 2

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

A. Observations of the Internal Effect of Refactoring

This section shows the effect of the refactoring patterns on the internal quality metrics. The students were asked to use (-) to indicate a decrease, (+) to indicate an increase, and (0) to indicate no use or no change. Tables VIII and IX display the effect of refactoring on the internal quality attributes as reported by both teams.

B. Number of Times Applying the Refactoring Patterns

Tables X and XI show the number of refactoring patterns that have been applied by both teams in each iteration.

X. INTERVIEW RESULTS

Subsequent to finishing the project, the consequences of the ANP assessment for ranking the refactoring patterns were appeared to the members so as to lead the interviews. Not all outcomes were as expected and a few results were surprising. The interviews included open-ended inquiries so as to gather the members' viewpoints about the ANP, their points of view on its advantages and disadvantages in XP, too to gather their perspectives about the best application for ANP in XP among all specified practices. The gathered information was contained written by hand notes from the interviews.

The interview findings show positive remarks from the members with respect to the ANP. The ANP was a useful approach in explaining struggle points of view, and urged each colleague to take an interest in deciding. The fundamental concern was the time it took amid the ANP assessment, and the quantity of pairwise comparisons. Another suggestion was applying the ANP in more XP practices and concentrate the impacts. All ANP colleagues recommended applying ANP in their future XP projects.

On the other hand, Team 2 was not totally happy with the procedure of their decisions. A portion of the colleagues complained about that the most experienced member had more voting weight than others, which lead them to take after decisions that they dislike. Another issue is that the ANP enabled us to know the distinction between each ranking position in a rate; in any case, Team 2 couldn't determined the measure of contrast between each ranked pattern and criterion.

XI. QUESTIONNAIRES

Surveys were dispersed among the members keeping in order to gather their experiences and perspectives. The given surveys comprised of two areas. The principal area included inquiries concerning ANP as a ranking and decision tool, for example, catching the required data, decency of the decision structure, clearness of criteria included, and clearness of alternatives included. The second area included inquiries regarding the advantages of every XP practice, and the students' fulfillment, for example, improving the group correspondence, elucidating the ranking issue, making positive discourse and learning chances, group performance, and fulfillment of the last consequences of the ANP. In this study, a seven-point Likert scale was used to decide the worthiness level of the ANP approach as follows:

1) Totally unacceptable.
2) Unacceptable.
3) Slightly unacceptable.
4) Neutral.
5) Slightly acceptable.
6) Acceptable.
7) Perfectly Acceptable.

In the wake of finishing the questionnaire, the same steps were followed as in [32]with a specific end goal to total the gathered information and show the aggregate agreeableness rate.

TABLE X. NUMBER OF REFACTORING PATTERNS WAS APPLIED BY TEAM 1 IN EACH ITERATION

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Total
Extract Class	0	7	6	4	5	22
Inline Class	0	4	2	3	5	14
Rename Method	0	42	38	24	7	101
Pull Up Method	0	0	0	0	0	0
Extract Method	0	19	13	11	16	59

TABLE XI. NUMBER OF REFACTORING PATTERNS WAS APPLIED BY TEAM 2 IN EACH ITERATION

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Total
Extract Class	0	3	5	2	6	16
Inline Class	0	2	0	0	0	2
Rename Method	0	3	4	4	6	17
Pull Up Method	0	1	0	4	1	6
Extract Method	0	4	3	2	3	12

The total acceptability percentage can be obtained as follows:
The total acceptability percentage (TAP)= the average score x $\frac{100}{7}$.
Where the average score = the sum of all scores given by team members / number of the team members.

The following rates show the worthiness level of the ANP as a ranking and decision tool:

- Enhancing team communication: 82%.
- Maximizing team performance: 87%.
- Supporting positive discussion and learning chances: 72%.
- Clearing up conflict perspectives among the team members: 87%.
- Defining the ranking problem: 91%.
- Satisfaction of the ANP final results 71%.

From various information sources, the information was gathered. By contrasting the gathered information and the study propositions in view of the understanding of the criteria that were specified above, we will investigation this gathered information. The followings are the study propositions and their answers:

- For the first proposition, we can see that both the alternatives and criteria are organized adequately, and considered in Fig. 3. Likewise, the final results and targets of the ANP use in ranking the refactoring techniques can be found in Table III, which showed the ranking of the ANP team for the XP refactoring patterns, and extract method was ranked as the highest.
- The survey statement 'satisfaction of the ANP final outcomes' supported the second proposition, and the comment of this was positive, which is 71 %. In addition, the statement ' clearing up conflict points of view among the developers' supported the third proposition, and the score was 87 %.

XII. VALIDITY

In this part, related threats to the validity are clarified. These threats are construct validity, external validity, internal validity, and reliability. Several studies underscored that case studies are hard to analyze because of biases and validity threats as described in [33] empirical studies in general and case studies specifically are inclined to predispositions and validity threats that make it hard to control the nature of the study in order to generalize its outcomes [33].

A. Construct Validity

Construct validity guarantees that the treatment mirrors the develop of the reason well, and the result mirrors the construct of the impact well [34]. It manages coordinating the idea being inquired about and considered, to the particular measurements. The modest number of members is the fundamental risk to this case study.

Using different strategies to guarantee the validity of the outcomes decreased this threat. Some of these strategies are:

- Data triangulation: a noteworthy favorable position of case study is the chance to utilize different sources of proof [35]. A proof chain is built through using interviews and questionnaire with different sorts of members with various abilities and experience levels, and the utilization of members' remarks and numerous perceptions. Hence, a valid conclusion can be come to.
- Methodological triangulation: employing a combination of research techniques such as organizing an XP project to serve the study purpose, questionnaire, findings of ANP pairwise comparisons, researchers' notes, and interviews.
- Member checking: showing the final results to the members is recommended. This issue was addressed by showing the final findings to all participants in order to ensure the study accuracy and to avoid researcher bias.

B. Internal Validity

Internal validity is tied in with ensuring the result is caused by the treatment (the impact). This kind of validity is just identified with explanatory case study. This issue might be tended to by connecting all information sources with respect to the research questions, and connecting the research questions to the study propositions.

C. External Validity

External validity guarantees the connection between the construct and the impact to ensure that the study will be generalized to a different environment [34]. In this investigation, extra case study will be need to be conducted in various situations, for example, industry to include more specialists from the field. Leading such a case study will help in looking at the different outcomes and discoveries from various conditions. Future work will add to expanded External validity.

D. Reliability

Reliability deals with the procedure of the gathered data and results. Similar conclusions and findings can be arrived by different researchers when following the same procedure, and using the same data. This might be done through the availability of same research questions, data collection, and case studies designed by other researchers.

XIII. Conclusion

After using the analytic network process to rank the refactoring patterns used in extreme programming, ANP was an appropriate and beneficial tool that gave the development team a good understanding for determining the most valuable refactoring patterns. The participants evaluated various refactoring patterns based on four internal code quality attributes, which were complexity, cohesion, code-size, and coupling. The most refactoring patterns that have enhanced the code quality in our study were Extract Method and Extract Class. in addition, the other mentioned refactoring patterns have added advantages to the code quality as well. Moreover, the ANP allowed us to specify the difference between each element in our model by a percentage, while the traditional XP team were not be able to do that. The ANP helped the team members resolve conflicts based on a structured approach grounded in scientific principles. The ANP ended up simplifying decision making, which maximized the effect of the software being developed. Team 1 members reconciled their conflicts of perspectives based on a mathematical approach. This maximized their satisfaction with the team's decisions.

Acknowledgement

Aljuhani's research is supported by the Saudi Cultural Bureau in Canada and Taibah University.

References

[1] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[2] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 576–585.

[3] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[4] J. Simmonds and T. Mens, "A comparison of software refactoring tools," *Programming Technology Lab*, 2002.

[5] E. Murphy-Hill, A. P. Black, D. Dig, and C. Parnin, "Gathering refactoring data: a comparison of four methods," in *Proceedings of the 2nd Workshop on Refactoring Tools*. ACM, 2008, p. 7.

[6] R. Marticorena, C. López, J. Pérez, and Y. Crespo, "Assisting refactoring tool development through refactoring characterization," in *Proceedings of the 6th International Conference on Software and Data Technologies*, vol. 2, 2011.

[7] S. Counsell, Y. Hassoun, G. Loizou, and R. Najjar, "Common refactorings, a dependency graph and some code smells: an empirical study of java oss," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 288–296.

[8] E. Murphy-Hill, "A model of refactoring tool use," *Proc. Wkshp. Refactoring Tools*, 2009.

[9] D. Roberts, J. Brant, and R. Johnson, "A refactoring tool for smalltalk," *Urbana*, vol. 51, p. 61801, 1997.

[10] M. Katić and K. Fertalj, "Towards an appropriate software refactoring tool support," in *WSEAS international conference on applied computer science*, 2009, pp. 140–145.

[11] J. Mahmood and Y. R. Reddy, "Usability of refactoring tools for java development," in *Proceedings of 1st Indian Workshop on Reverse Engineering, co-located with 3rd India Software Engineering Conference (ISEC2010) Feb*, 2010.

[12] S. Hayashi, M. Saeki, and M. Kurihara, "Supporting refactoring activities using histories of program modification," *IEICE transactions on information and systems*, vol. 89, no. 4, pp. 1403–1412, 2006.

[13] T. L. Saaty, "Fundamentals of the analytic network processdependence and feedback in decision-making with a single network," *Journal of Systems science and Systems engineering*, vol. 13, no. 2, pp. 129–157, 2004.

[14] T. L Saaty, "The analytic network process," *Iranian Journal of Operations Research*, vol. 1, no. 1, pp. 1–27, 2008.

[15] T. L. Saaty, "Decision making with the analytic hierarchy process," *International journal of services sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[16] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2013.

[17] A. Aljuhani, L. Benedicenti, and S. Alshehri, "Ranking xp prioritization methods based on the anp," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 5, pp. 1–8, 2017.

[18] A. Bustamante and R. Sawhney, "Agile xxl: Scaling agile for project teams, seapine software, inc," 2015.

[19] V. Lalsing, S. Kishnah, and S. Pudaruth, "People factors in agile software development and project management," *International Journal of Software Engineering & Applications*, vol. 3, no. 1, p. 117, 2012.

[20] B. Rumpe and P. Scholz, "Scaling the management of extreme programming projects," *arXiv preprint arXiv:1409.6604*, 2014.

[21] S. Ambler, "Agile teams making decisions: Decision making tools," http://www.ambysoft.com/surveys/success2010.html, accessed: 2015-02-24.

[22] L. Zhao and J. H. Hayes, "Rank-based refactoring decision support: two studies," *Innovations in Systems and Software Engineering*, vol. 7, no. 3, pp. 171–189, 2011.

[23] J. Al Dallal and L. C. Briand, "A precise method-method interaction-based cohesion metric for object-oriented classes," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 2, p. 8, 2012.

[24] H. A. Sahraoui, R. Godin, and T. Miceli, "Can metrics help bridging the gap between the improvement of oo design quality and its automation," in *Proceedings of the International Conference on Software Maintenance, ICSM*, 2000.

[25] E. Stroulia and R. Kapoor, "Metrics of refactoring-based development: An experience report," in *OOIS 2001*. Springer, 2001, pp. 113–122.

[26] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi, "Does refactoring improve reusability?" in *International Conference on Software Reuse*. Springer, 2006, pp. 287–297.

[27] B. Du Bois and T. Mens, "Describing the impact of refactoring on internal program quality," in *International Workshop on Evolution of Large-scale Industrial Software Applications*, 2003, pp. 37–48.

[28] K. O. Elish and M. Alshayeb, "Using software quality attributes to classify refactoring to patterns." *JSW*, vol. 7, no. 2, pp. 408–419, 2012.

[29] J. Ratzinger, M. Fischer, and H. Gall, *Improving evolvability through refactoring*. ACM, 2005, vol. 30, no. 4.

[30] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 576–585.

[31] "The software design metrics tool for the uml," https://www.sdmetrics.com/DProp.html, accessed: 2017-04-12.

[32] S. Alshehri and L. Benedicenti, "Ranking approach for the user story prioritization methods," *J Commun Comput*, vol. 10, pp. 1465–1474, 2013.

[33] R. Lincke, M. Höst, and P. Runeson, "How do phd students plan and follow-up their work?–a case study," *School of Mathematics and Systems Engineering, University Sweden*, 2007.

[34] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[35] R. Yin, "Case study research design and methods 3rd ed sage publications," 2002.