RANKING REFACTORING PATTERNS USING THE ANALYTICAL HIERARCHY PROCESS

Eduardo Piveta[1], Ana Moreira[2], Marcelo Pimenta[1]
João Araújo[2], Pedro Guerreiro[3] and R. Tom Price[1]

[1] *Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil*

[2] *Departamento de Informática, Universidade Nova de Lisboa, Caparica, Portugal*

[3] *Departamento de Engenharia Electrónica e Informática, Universidade do Algarve, Faro, Portugal*

Keywords: Refactoring, Refactoring Opportunities, Analytical Hierarchy Process.

Abstract: This paper describes how to rank refactoring patterns to improve a set of quality attributes of a piece of software. The Analytical Hierarchy Process (AHP) is used to express the relative importance of the quality attributes and the relative importance of refactoring patterns in regards to those selected quality attributes. This ranking of refactoring patterns can be used to focus the refactoring effort on the most beneficial patterns to the software being developed or maintained.

1 INTRODUCTION

Refactoring (Opdyke, 1992; Fowler et al., 1999; Mens and Tourwe, 2004) is the process of improving the design of software systems without changing its externally observable behaviour. Refactoring can help to incrementally improve the quality attributes of a software system through the application of behavioural preserving transformations called refactoring patterns.

When refactoring a software module, the developer has to correctly evaluate the trade-offs between refactoring patterns in terms of the affected quality attributes. As these quality attributes can be conflicting with each other (Boehm and In, 1996), the task of selecting optimal refactoring patterns can be hard. As the number of possible places to refactor can be very high (Fowler et al., 1999), the developer has to narrow the search for refactoring opportunities to apply refactoring patterns that bring benefits in terms of the expected quality attributes.

Current research on the identification of refactoring opportunities (Bois and Mens, 2003; Mens et al., 2005; Bois, 2006) focuses on the improvements of quality attributes considering each *individual* application of a refactoring pattern, but does not consider how this search can be narrowed to only those patterns that improve the desired quality attributes of a software system. Currently, there are no automated mechanisms to rank refactoring patterns in terms of quality attributes.

In this paper, we propose an approach to rank refactoring patterns in terms of a set of quality attributes. The relative importance of quality attributes over the others and the relative importance of refactoring patterns over quality attributes is quantitatively expressed using the Analytical Hierarchy Process (AHP) multi-criteria decision method (Saaty, 1990; Saaty, 2003).

Using this quantified information, a ranking of refactoring patterns in terms of quality attributes can be automatically computed. The use of such ranking can optimise the search for refactoring opportunities, enabling the developer to focus on the refactoring patterns that improve the quality attributes most.

The creation of a refactoring patterns ranking is exemplified using three quality attributes and four refactoring patterns. Although the example is based on refactoring patterns for object-oriented software, the approach can be used in other paradigms. Moreover, the approach is general enough to be successfully applied in software models, source code or other artefacts.

This paper is organized as follows. Section 2 details AHP. Section 3 shows how to rank refactoring patterns according to a set of quality attributes using AHP and describes tool support. Section 4 discusses related work and Section 5 concludes the paper.

Piveta E., Moreira A., Pimenta M., Araújo J., Guerreiro P. and Tom Price R. (2008).
RANKING REFACTORING PATTERNS USING THE ANALYTICAL HIERARCHY PROCESS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 195-200
DOI: 10.5220/0001686001950200

2 ANALYTICAL HIERARCHY PROCESS

The Analytical Hierarchy Process (AHP) (Saaty, 1990) is a decision making method for evaluating a set of different alternative solutions of a given problem. It focuses on finding an optimal solution using qualitative and quantitative decision analysis.

AHP comprises of five steps: (a) definition of the problem and its objective, (b) hierarchical representation, (c) estimation of priorities, (d) synthesis and (e) results consistency analysis.

The *definition of the problem and its objective* establishes the context in which the decision making process will occur. Next, the problem is *hierarchically represented*, with the objective having several associated criteria and each criterion several alternatives.

In the *priorities estimation* step, the developer defines the priorities for the criteria and alternatives. The relative importance of each criterion over the others and each alternative over the others is ascertained using pairwise comparisons. The scale in Table 1 is used to numerically express the relative importance between the criteria and the alternatives.

Table 1: The numerical values of each relative importance (Saaty, 1990).

Value	Relative Importance
1	Same importance
2	Slightly more important
3	Weakly more important
4	Weakly to moderately more important
5	Moderately more important
6	Moderately to strongly more important
7	Strongly more important
8	Greatly more important
9	Absolutely more important

In this process, a pairwise matrix can be created, containing the relative importance of the criteria over the others. Consider, for example, a set of criteria $\mathcal{C} = \{c_1, \ldots, c_n\}$ and a matrix $\mathcal{M}$ representing the relative importance of one criterion over another $W = \{w_{11}, w_{12}, \ldots, w_{nn}\}$. The pairwise matrix in this case, is constructed as follows:

$$\mathcal{M} = \begin{bmatrix} 1 & w_{12} & \cdots & w_{1n} \\ 1/w_{21} & 1 & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1/w_{n1} & 1/w_{n2} & \cdots & 1 \end{bmatrix} \quad (1)$$

For example, consider three criteria c_1, c_2, c_3, where c_1 is *weakly more important* than c_3, c_2 is *slightly more important* than c_1 and c_2 is *weakly to moderately more important* than c_3. A pairwise matrix $\mathcal{M}$ containing the importance of each criterion over another can be created as follows:

$$\mathcal{M} = \begin{array}{c} \overbrace{\begin{array}{ccc} c_1 & c_2 & c_3 \end{array}} \\ \begin{bmatrix} 1 & 1/2 & 3 \\ 2 & 1 & 4 \\ 1/3 & 1/4 & 1 \end{bmatrix} \end{array} \begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} \quad (2)$$

The same process is done to the alternatives. For each criterion, a set of pairwise comparisons are created between the alternatives and a pairwise matrix is created to represent these comparisons.

The next step of AHP, named *synthesis*, computes a vector containing the relative weights of all the criteria in the matrix and a vector for each alternative matrix. To compute each vector, the respective matrix is squared successively. In each iteration, the row sums are calculated and normalized. The computation stops when the differences of these sums in two consecutive calculations are smaller than a prescribed value. Usually two to four iterations are sufficient.

For the example matrix $\mathcal{M}$, the ranking of priorities can be derived from the weights vector of the matrix. To compute this vector, the matrix is squared, the sum of the row values is computed and the values are normalized:

$$\overbrace{\begin{bmatrix} 3.00 & 1.75 & 8.00 \\ 5.33 & 3.00 & 14.00 \\ 1.17 & 0.67 & 3.00 \end{bmatrix}}^{squared\ matrix} = \overbrace{\begin{bmatrix} 12.7500 \\ 22.3332 \\ 4.8333 \end{bmatrix}}^{sum\ of\ rows}$$

$$\textbf{Total} \quad 39.9165$$

$$Eigenvector = \overbrace{\begin{bmatrix} 0.3194 \\ 0.5595 \\ 0.1211 \end{bmatrix}}^{normalized}$$

$$\textbf{Norm. Total} \quad 1.0000$$

Iterating the process again, the computed vector is:

$$V' = \langle 0.3196 \ \ 0.5584 \ \ 0.1220 \rangle$$

Additional iterations do not change these values (using four digits precision). This normalized vector is called the principal eigenvector (Saaty, 2003) of the matrix. To compute the overall ranking of alternatives in terms of the selected criteria, a matrix whose columns are the alternatives eigenvectors is multiplied by the criteria eigenvector. The resulting vector is the overall ranking.

The last step of AHP, *results consistency analysis*, evaluates the consistency level of the pairwise comparison matrix. More details on how to compute a consistency ratio are described by Saaty (Saaty, 1990).

3 RANKING REFACTORING PATTERNS

This section describes the main steps for ranking refactoring patterns according to a set of preferred quality attributes and exemplifies each step using three quality attributes and four refactoring patterns, showing how AHP can be used to create a ranking of refactoring patterns in terms of quality attributes.

3.1 Overview

Starting with a set of candidate refactoring patterns and a set of selected quality attributes, the developer performs the following steps to generate a ranking of refactoring patterns according the preferred quality attributes:

1. *Create pairwise comparisons for quality attributes:* The first step is to create pairwise comparisons between the selected quality attributes. This is the developer's main task in this approach, as the relationship of quality attributes is usually specific to a project.
2. *Create pairwise comparisons for refactoring patterns:* A tool provider can make available a knowledge base containing pairwise comparisons for typical refactoring patterns and quality attributes. The developer can then retrieve the pairwise comparisons from this base or add new ones to the knowledge base (if there are no pairwise comparisons available for a particular refactoring pattern or quality attribute).
3. *Compute the quality attributes ranking:* In this task, a pairwise matrix is created to express the relationship between the quality attributes. The quality attributes ranking is the eigenvector of the quality attributes pairwise matrix. This step and the next ones can be automated.
4. *Compute the refactoring patterns rankings:* For each quality attribute, a pairwise matrix is created to quantitatively express the relationship of the refactoring patterns vs. the quality attribute. The ranking of refactoring patterns considering each quality attribute in isolation is the eigenvector of the respective pairwise matrix.
5. *Compute the overall ranking:* A ranking of the selected refactoring patterns is computed using the quality attributes eigenvector and the alternatives (refactoring patterns) eigenvectors. To compute the ranking of the refactoring patterns given the set of quality attributes, a matrix is created with the criteria (the quality attributes) and the alternatives (the refactoring patterns). This matrix is multiplied by the eigenvector of the quality attributes pairwise matrix.

This overall ranking can be used to focus the search for refactoring opportunities for the best ranked refactoring patterns, instead of looking for refactoring opportunities for refactoring patterns that contribute little to the overall software quality (i.e. are low ranked). Once a knowledge base containing the relationship between the refactoring patterns and the quality attributes is created, the developer has only to provide pairwise comparisons for the quality attributes, as the matrix manipulation activities can be automated.

3.2 Creating the Quality Attributes Pairwise Comparisons

The importance of the selected quality attributes depends on the values of the development team, the process, the project and the organisation. Each project can have different sets and ordering of quality attributes.

In this example, the following quality attributes are considered: reusability, simplicity and comprehensibility. The following possible judgments for the selected quality attributes are expressed as pairwise comparisons:

- Simplicity is *moderately more important* than reusability and *slightly more important* than comprehensibility;
- Comprehensibility is *slightly more important* than reusability.

Note that these pairwise comparisons are hypothetical. Each project can have different needs in terms of quality attributes and can have different weights for each quality attribute. The responsibility for creating these comparisons can be delegated to a quality analyst, made by the project leader or by other means.

3.3 Creating the Refactoring Patterns Pairwise Comparisons

For the example being developed, four different refactoring patterns were chosen, because they manipulate classes, methods and interfaces: *Pull Up Method* (pm), *Rename Class* (rc), *Inline Method* (im), and *Extract Interface* (ei). They are compared and ranked in terms of the selected quality attributes.

The first step is evaluating each refactoring pattern in terms of each quality attribute, considering how much one refactoring pattern improves each quality attribute compared to the others. Let us suppose that the developer created a set of pairwise comparisons of the refactoring patterns in terms of the first quality attribute (simplicity), as follows:

- Pull Up Method is *strongly more important* than Rename Class, *weakly more important* than Inline Method and *strongly more important* than Extract Interface
- Inline Method is *slightly more important* than Rename Class and *strongly more important* than Extract Interface
- Rename Class is *weakly more important* than Extract Interface

The second quality attribute is reusability, for which the following judgments are made to exemplify the process:

- Pull Up Method is *absolutely more important* than Rename Class, *moderately to strongly more important* than Inline Method and *slightly more important* than Extract Interface
- Inline Method has the *same importance* than Rename Class
- Extract Interface is *strongly more important* than Rename Class and Inline Method

The third quality attribute is comprehensibility. For this quality attribute the following judgments are made using pairwise comparisons:

- Extract Interface is *slightly more important* than Pull Up Method and Rename Class and it is *absolutely more important* than Inline Method;
- Rename Class is *slightly more important* than Pull Up Method and *strongly more important* than Inline Method;
- Pull Up Method is *moderately more important* than Inline Method

These pairwise comparisons can be inserted in a knowledge base to enable future reuse of the relations between refactoring patterns and quality attributes.

3.4 Computing the Quality Attributes Ranking

Using the pairwise comparisons defined in the previous section and the numerical values on Table 1, the quality attributes pairwise matrix $\mathcal{Q}$ is straightforwardly created from the defined pairwise comparisons:

$$\mathcal{Q} = \begin{array}{cccc} simp. & reus. & comp. & \\ \hline 1.00 & 5.00 & 2.00 & simp. \\ 0.20 & 1.00 & 0.50 & reus. \\ 0.50 & 2.00 & 1.00 & comp. \end{array}$$

This pairwise matrix is used to compute the weight of each quality attribute. In this matrix, the computed eigenvector $\mathcal{E}_q$ is:

$$\mathcal{E}_q = \begin{array}{cc} 0.5954 & simp. \\ 0.1283 & reus. \\ 0.2764 & comp. \end{array}$$

The $\mathcal{E}_q$ vector shows that the most important quality attribute in this setting is simplicity, then comprehensibility and last reusability. This setting can vary from project to project. The computed weights for each quality attribute define which are the preferred refactoring patterns for the selected quality attributes. In this case, the consistency ratio of the $\mathcal{Q}$ matrix is 0.48% (the matrix is consistent).

3.5 Computing the Refactoring Patterns Ranking

The ranking of refactoring patterns for each quality attribute and the ranking of quality attributes can be automatically created as follows.

The Simplicity Ranking. First, the pairwise comparisons for the *simplicity* quality attribute are translated to their numerical equivalents and a pairwise matrix $\mathcal{S}$ is computed as follows:

$$\mathcal{S} = \begin{array}{ccccc} pm & im & rc & ei & \\ \hline 1.00 & 3.00 & 7.00 & 7.00 & pm \\ 0.33 & 1.00 & 2.00 & 7.00 & im \\ 0.14 & 0.50 & 1.00 & 3.00 & rc \\ 0.14 & 0.14 & 0.33 & 1.00 & ei \end{array}$$

Here is the computed eigenvector of $\mathcal{S}$, named $\mathcal{E}_s$:

$$\mathcal{E}_s = \begin{array}{cc} 0,593 & pm \\ 0,245 & im \\ 0,113 & rc \\ 0,050 & ei \end{array}$$

In this case, Pull Up Method is the preferred refactoring pattern to be used, in terms of simplicity, when compared with the other three refactoring patterns (in order): Inline Method, Rename Class and Extract Interface. The consistency ratio is 5.83% and the pairwise matrix is consistent. Note that the difference between them is high. The developer can focus on those patterns with high values of relative importance for the quality attribute.

The Reusability Ranking. The reusability pairwise matrix $\mathcal{R}$, after translating the pairwise comparisons to their numerical equivalents, is:

$$\mathcal{R} = \begin{array}{c} \overbrace{\begin{array}{cccc} pm & im & rc & ei \end{array}} \\ \begin{bmatrix} 1.00 & 6.00 & 9.00 & 2.00 \\ 0.17 & 1.00 & 1.00 & 0.14 \\ 0.11 & 1.00 & 1.00 & 0.14 \\ 0.50 & 7.00 & 7.00 & 1.00 \end{bmatrix} \end{array} \begin{array}{c} pm \\ im \\ rc \\ ei \end{array}$$

The computed eigenvector $\mathcal{E}_r$ is:

$$\mathcal{E}_r = \begin{bmatrix} 0.522 \\ 0.063 \\ 0.056 \\ 0.358 \end{bmatrix} \begin{array}{c} pm \\ im \\ rc \\ ei \end{array}$$

Considering the ranking of refactoring patterns for reusability, the preferred refactoring pattern is Pull Up Method, followed by Extract Interface. The Inline Method and Rename Class refactoring patterns do not have much impact on this quality attribute. The consistency ratio is 2.52% and the pairwise matrix is considered consistent.

The Comprehensibility Ranking. The pairwise matrix $\mathcal{C}$ for comprehensibility is also created:

$$\mathcal{C} = \begin{array}{c} \overbrace{\begin{array}{cccc} pm & im & rc & ei \end{array}} \\ \begin{bmatrix} 1.00 & 5.00 & 0.50 & 0.50 \\ 0.20 & 1.00 & 0.14 & 0.11 \\ 2.00 & 7.00 & 1.00 & 0.50 \\ 2.00 & 9.00 & 2.00 & 1.00 \end{bmatrix} \end{array} \begin{array}{c} pm \\ im \\ rc \\ ei \end{array}$$

And the computed eigenvector $\mathcal{E}_c$ is:

$$\mathcal{E}_c = \begin{bmatrix} 0.196 \\ 0.044 \\ 0.304 \\ 0.457 \end{bmatrix} \begin{array}{c} pm \\ im \\ rc \\ ei \end{array}$$

Considering only comprehensibility, the best ranked refactoring pattern is Extract Interface, followed by Rename Class and Pull Up Method. The Inline Method refactoring pattern does not have much impact in terms of comprehensibility compared to the other ones selected. The consistency ratio is 1.9% (the pairwise matrix is consistent). Note that the order of the refactoring patterns is different, depending on the quality attribute.

3.6 Computing the Overall Ranking

For the example used in this paper the ranking is computed by:

$$\mathcal{O} = \begin{array}{c} \overbrace{\begin{array}{ccc} simp. & reus. & compre. \end{array}} \\ \begin{bmatrix} 0.593 & 0.522 & 0.196 \\ 0.245 & 0.063 & 0.044 \\ 0.113 & 0.056 & 0.304 \\ 0.050 & 0.358 & 0.457 \end{bmatrix} \end{array} * \begin{array}{c} \overbrace{criteria} \\ \begin{bmatrix} 0.5954 \\ 0.1283 \\ 0.2764 \end{bmatrix} \end{array}$$

Considering the initial quality attributes, the pairwise comparisons between them and the judgments for the alternatives, the ranking $\mathcal{O}$ of refactoring patterns is:

$$\mathcal{O} = \begin{bmatrix} 0.4743 \\ 0.1658 \\ 0.1583 \\ 0.2018 \end{bmatrix} \begin{array}{c} pm \\ im \\ rc \\ ei \end{array}$$

The overall ranking shows that, considering the selected quality attributes, the selected refactoring patterns and the pairwise comparisons made, the best ranked refactoring pattern is Pull Up Method, followed by Extract Interface, Inline Method and Rename Class.

After the ranking is created, the developer can define a threshold to reduce the initial set of refactoring patterns to only those that have a ranking value higher than this threshold. He can decrease the threshold value to add more refactoring patterns to the search for refactoring opportunities or increase it if the search for low ranked refactoring patterns is not being fruitful. The use of a threshold narrows the search for refactoring opportunities, focusing on the best ranked refactoring patterns.

3.7 Tool Support

A proof-of-concept tool was developed to assess the practical use of the proposed approach. It (i) converts pairwise matrices expressing the relative importance of each quality attribute over the others and of each refactoring pattern over the others (for each quality attribute) to the equivalent AHP numerical representation, (ii) creates the pairwise matrices, (iii) computes the eigenvectors, (iv) elaborates the quality attributes ranking and (vi) computes the rankings of refactoring patterns regarding each quality attribute.

These rankings are used to automatically compute the overall ranking. If the refactoring patterns matrices are created by a tool provider, for example, the user has only to provide the pairwise comparisons between the quality attributes.

Note that the ranking does not need to be updated frequently. Once the matrices relating refactoring patterns to quality attributes are created (by a tool provider, for example), the developer only has to inform the pairwise comparisons for the relative importance of each quality attribute over the others.

The ranking must be recalculated if one of the change scenarios occur: changes in the pairwise comparisons; addition of a refactoring pattern; addition of a quality attribute. The tool can recalculate the ranking automatically.

4 RELATED WORK

Du Bois and Mens (Bois and Mens, 2003), (Bois, 2006) suggest the evaluation of refactoring patterns by identifying conditions in which their application minimize coupling and maximize cohesion. Their formal analysis can be used together with our approach to provide additional information for the developer to express the relative importance of refactoring patterns over the quality attributes using pairwise comparisons.

Tourwe and Mens (Tourwe and Mens, 2003) use logic meta programming to identify refactoring opportunities in a software design and propose the application of refactoring patterns. Their approach can be used together with ours to improve the results by focusing on the refactoring patterns that improve most the set of quality attributes selected by the developers.

Mens et al. (Mens et al., 2003) state that an open problem is to assess the effects of a refactoring pattern on the software quality, as some refactoring patterns remove redundancy, raise abstraction or modularity level and others have negative impact on reusability, for example. By classifying refactoring patterns in terms of the quality attributes they affect, the effect of a refactoring on the software quality can be estimated. In this paper, we provide a quantitative approach to rank a set of refactoring patterns according to the quality attributes that the developers are concerned about.

5 CONCLUSIONS

Usually, there is room for improvements in existing software projects. However, resources are finite and must be directed to those activities that bring more benefits to the project. Considering refactoring activities, the developers should focus on the search for refactoring opportunities for those refactoring patterns that are more likely to improve the software being developed or maintained.

AHP can help the developers to express the relationship between quality attributes and refactoring patterns and quality attributes between themselves. These relations are used to compute a ranking of refactoring patterns (according to the selected quality attributes), which can be used to focus the effort of searching for refactoring opportunities for those refactoring patterns that can have more impact in the software quality.

Without focusing in a restricted set of refactoring patterns, the developers can be losing time with refactoring opportunities that bring little or nothing to the software quality. The approach described in this paper is adaptable: the quality attributes, the refactoring patterns and the weights can be changed and a new ranking computed automatically.

ACKNOWLEDGEMENTS

This work has been partially supported by CNPq under grant No. 140046/2006-2 for Eduardo Piveta, by Capes-Grices grant No. 166/06 and by PROSUL No. 490478/2006-9 - Latin-America Research Network on Aspect-Oriented Software Development (Latin- AOSD).

REFERENCES

Boehm, B. W. and In, H. (1996). Identifying quality-requirement conflicts. *IEEE Software*, 13(2):25–35.

Bois, B. D. (2006). *A Study of Quality Improvements by Refactoring*. PhD thesis, Universiteit Antwerpen.

Bois, B. D. and Mens, T. (2003). Describing the impact of refactorings on internal program quality. In *1st Workshop on Evolution of Large-scale Industrial Software Applications - ELISA'03. Amsterdam, Netherlands.*

Fowler, M., Beck, K., Brant, J., Opdyke, W. F., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Mens, T., Demeyer, S., Bois, B. D., Stenten, H., and van Gorp, P. (2003). Refactoring: Current research and future trends. *ENTCS - Elsevier*, 82(3):483 – 499.

Mens, T., Taentzer, G., and Runge, O. (2005). Detecting structural refactoring conflicts using critical pair analysis. *ENTCS - Elsevier*, 127(3):113 – 128.

Mens, T. and Tourwe, T. (2004). A survey of software refactoring. *IEEE Trans. on Software Engineering*, 30(2):126 – 139.

Opdyke, W. F. (1992). *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois.

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research - Elsevier*, 48(1):9 – 26.

Saaty, T. L. (2003). Decision-making with the ahp: Why is the principal eigenvector necessary? *European Journal of Operational Research - Elsevier*, 145(1):85 – 91.

Tourwe, T. and Mens, T. (2003). Identifying refactoring opportunities using logic meta programming. In *7th European Conf. on Software Maintenance and Reengineering - CSMR'03. Benevento, Italy.*, pages 91 – 100.