Recommendation System for Design Patterns in Software Development: An DPR Overview

Francis Palma, Hadi Farzin, Yann-Gaël Guéhéneuc
École Polytechnique de Montréal
Montréal, Canada
{*francis.palma, hadi.farzin, yann-gael.gueheneuc*}*@polymtl.ca*

Naouel Moha
Université du Québec à Montréal
Montréal, Canada
moha.naouel@uqam.ca

***Abstract*—Software maintenance can become monotonous and expensive due to ignorance and misapplication of appropriate design patterns during the early phases of design and development. To have a good and reusable system, designers and developers must be aware of large information set and many quality concerns, e.g., design patterns. Systems with correct design pattern may ensure easy maintenance and evolution. However, without assistance, designing and development of software systems following certain design patterns is difficult for engineers. Recommendation systems for software engineering can assist designers and developers with a wide range of activities including suggesting design patterns. With the help of pattern recommenders, designers can come up with a reusable design. We provide a *Design Pattern Recommender* (DPR) process overview for software design to suggest design patterns, based on a simple Goal-Question-Metric (GQM) approach. Our prototype provides two-fold solution. In the primary-level, DPR only proposes one or more design patterns for a problem context, and in the secondary level, for a initial set of design, DPR refactors models and suggests design patterns. Our preliminary evaluation shows that DPR has a good trade-off between accuracy and procedural complexity, comparing to other state-of-the-art approaches.**

***Keywords*-Recommendation system; Design pattern; Software reuse**

I. Introduction

Each pattern describes a problem that occurs over and over again in software development domain and then describes the core of the solution to that problem, in such a way that we can use this solution many times, without doing it the same way twice [3]. According to Gamma et al. [4], the use of design patterns, more precisely, of recurring design solutions for object-oriented systems, provides several advantages, from increased reusability and improved maintainability to comprehensibility of existing systems. Increasing number of available design patterns should lead to better quality, by proposing good solutions to recurring design problems. However, it is difficult to follow all new patterns during development and to choose the right patterns when faced with a design problem [7]. A general definition and description proposed by the organizers of the ACM International Conference on Recommender Systems states: *"Recommendation systems are software applications that aim to support users in their decision-making while interacting with large information spaces"*. For helping designers, a *recommendation system* can gather and analyze information and make decisions. The purpose of our work is two-fold. Firstly, recommend a design pattern for a specific problem context (primary-level) and, secondly, recommend one (or more) design pattern(s) suitable for a system with initial models already created by the designer (secondary-level). In this work, we will focus on the prototype of primary-level recommendation and keep the secondary level for future work. The remainder of this paper is organized as follows. Section II presents the literature of some design pattern recommendation techniques. Section III describes our approach, and Section IV, the design-prototype of the system. Section VI presents a preliminary evaluation. The concluding remarks and future work are discussed in Section VII.

II. Related Work

For suggesting design pattern, recommendation is also an important activity, to minimize designers' effort and helping them to have a good set of models with correct patterns. Thus, recommendation systems can play important role by recommending design patterns to overcome the information overload problem by exposing users to the most interesting items [5]. We discuss few recommendation approaches that have been proposed in the literature. Guéhéneuc et al. [6] introduced a simple recommender system, to help users in choosing among the 23 design patterns from the GoF [4], and the system was developed by analyzing the textual descriptions of design patterns. But complex design problems are not manageable, also no user feedback is associated with this approach, which makes it obsolete. Gomes et al. [9] proposed an approach that exploits user experience to suggest suitable patterns using the *ReBuilder* framework. ReBuilder adopts a case-based reasoning approach, where cases represent situations in which a pattern was already applied in the past in a software design. Here, cases are described in terms of class diagrams. But the main drawback of this approach is that such diagrams are not always available and does not produce relevance score. Kung et al. [8] proposed a prototype of an expert system for suggesting design patterns, that is closest to our work. With a wide range of patterns, narrowing down the selection through an interactive session (asking questions) with the users, Expert System for Suggesting Design Patterns (ESSDP) [8] effectively suggests design patterns. However, reducing the number of questions is still a challenge and more accuracy can be achieved. We also try to propose a similar approach

with the same goal, trying to minimize the number of questions, with a good relevance and accuracy.

III. The Methodology

In this section, we describe the methodology for Design Pattern Recommender (DPR). DPR is a selection type of expert system and hence we choose to use a ranking based selection approach for DPR. DPR is aimed to help designers to find or decide which pattern to use for a particular design problem. For developing an expert system like DPR consists of four steps, summarized as follows:

A. Identify the Circumstances in Which Patterns Can be Applied

Prior to constructing our knowledge base for DPR, we perform knowledge acquisition. Knowledge acquisition refers to the task of gathering the required knowledge and verifying it. Knowledge refers to in-depth study of the individual patterns, i.e., their properties, applicabilities, intents, examples, etc. During this phase, we review and analyze the literature on object-oriented software design patterns. Others may interview human experts to obtain knowledge on design patterns. We select design patterns by Gamma [4]. For example, From the description of the Adapter pattern [4], and following the guidelines suggested by Sommerville and Sawyer [1], we can derive the following set of conditions referring to some circumstances (in Figure 1, the child nodes of Adapter node are the intents of Adapter pattern):
C1: Need to convert interface.
C2: Want to solve issue of incompatible interfaces.

B. Refine the Circumstances With Sub-conditions

As a result of Step A, we obtain various trees similar to Figure 1 for each pattern. Here, we refine each condition into possible subconditions, i.e., from the intents as conditions to the applicabilities as sub-conditions for each pattern. In Figure 1, we can see that adapter pattern had two conditions C1 and C2 and then again it has three subconditions SC1, SC2 and SC3.

C. Formulate Questions to Ask Designers

Tree representation of knowledge about patterns obtained during Step A and B, are the means to derive questions to ask designers. From the answers to these questions, patterns receive weights given by designers. Here, we convert each node from the tree to questions. For example, for Adapter pattern, the first subcondition becomes 'Do you want to create a reusable class?' and the second subcondition 'Do you need to use several existing subclasses?'. The result of this step is a set of questions formulated for each tree generated from Step B.

D. Formulate GQM Model with the Defined Questions

We build a Goal-Question-Metric (GQM) [2] model for the interactivity. At top, we place the pattern names as the goals. In the question levels, we place 2-layered questions, where first layer represents conditions, followed by the subconditions. These bottom level questions will be asked directly to designers. Each of which will be responded with 'yes', 'no' or 'do not know', with a weight, assigned by the users. All the weights are positive integers between 1 and 9 (except 0 for 'do not know'). For example, any positive response will have impact on current pattern and any negative response will have impact on all the other patterns. Figure 2 shows a set of candidate design patterns, when designer chooses 'Architectural class' as the response of the level-0 question. In this figure, some other patterns are also presented that are not under 'Architectural' pattern class, as they are related. The calculation of total weight for each design patterns can formally be defined as in the Equation (1) below:

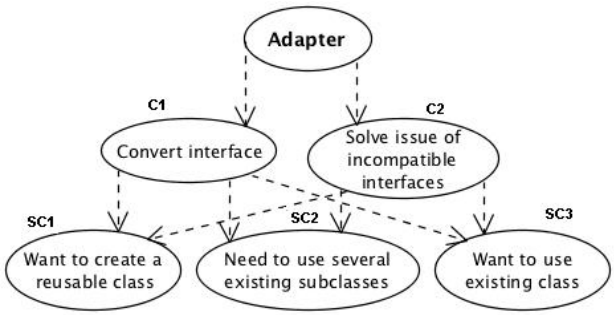

Figure 1. Adapter pattern with conditions (pattern-intents) and sub-conditions (pattern-applicabilities)

$$Pattern\_Name_{\text{Tot\_Weight}} = \forall_{\text{Applicability}} \left(\sum_{i=1}^{n} Weight\Big(Pattern\_Name_{\text{i\_Yes\_Score}}\Big) + \sum_{j=1}^{n} Weight\Big(Pattern\_Name_{\text{j\_No\_Score}}\Big) \right) \quad (1)$$

where, i $\neq$ j & i $\geq$ 1, j $\geq$ 1

In Equation (1), the $Pattern\_Name_{\text{Tot\_Weight}}$ is a function that calculates the total weight for a design pattern. $\sum_{i=1}^{n} Weight(Pattern\_Name_{\text{i\_Yes\_Score}})$ is a function that calculates the total weight for all the 'yes' responses for the current pattern and $\sum_{j=1}^{n} Weight(Pattern\_Name_{\text{j\_No\_Score}})$ returns the total weight for all the 'no' responses, calculated by the Weight function. We deploy this equation during interactive session for selecting the design pattern, as shown in Table I.

IV. Prototype

In this section, we provides a high-level overview of DPR system activities and components, resulting from the methodology described in Section III.

A. The DPR Knowledge Base (KB)

We have a central KB that plays an important role during the interactive session with the designer. We present an XML implementation of such a KB in Figure 3 that is a simple XML-based representation of the pattern specific properties and their relations with other patterns in the problem context. From Figure 3, we can see the root node of the XML DOM is *DesignPattern* followed by three children *DesignPatternType1 (2 and 3)*. Under the *DesignPatternType1*, the children are all the stand-alone patterns, each of which again have children, i.e., *Intents*, *Applicability* and *Link*. For the Link

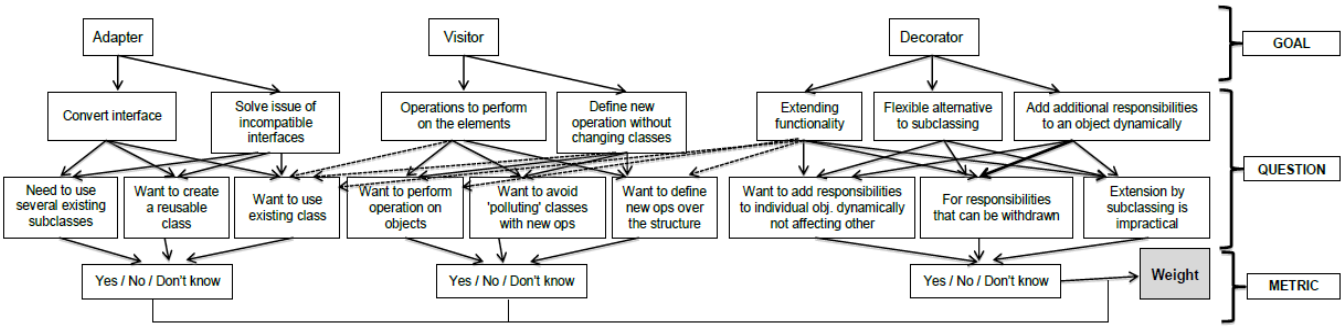

Figure 2. In DPR, a simple GQM model to select design pattern by asking questions

```
<?xml version="1.0" encoding="UTF-8"?>
<DesignPatternType id = "Architectural" description ="Class architecture related... ">
  <Patterns name = "Adapter">
    <Intent intent1 = "Convert interface" intent2 = "Solve issue of incompatible interfaces">
      <Applicability app1 = "Need to use several existing subclasses" app2 = "..." app3 = "...">
        <Link Pattern = "Visitor">
          <IncomingLink destitem = "Adapter.app3" srcitem = "Visitor.intent1"/>
          <IncomingLink destitem = "Adapter.app3" srcitem = "Visitor.intent2"/>
        </Link>
        <Link Pattern = "Decorator">
          <IncomingLink destitem = "Adapter.app3" srcitem = "Decorator.intent1"/>
        </Link>
      </Applicability>
    </Intent>
    <Values app1value = "Yes" app2value = "No" app3value = "Don't Know"/>
    <Weights app1weight = "" app2weight = "" app3weight = ""/>
  </Patterns>
</DesignPatternType>
```

Figure 3. XML-representation of DPR knowledge-base

node, it has another child node, i.e., *IncomingLink*. This node stores information for all the incoming relations to this pattern putting itself in the *destitem* and the pattern to which it is related to in *srcitem* nodes respectively.

B. DPR Process Model

Figure 4 shows the activity diagram for primary-level recommendation. According to Figure 4, we first ask users the top level (level 0) question for filtering the pattern type. Then, using the GQM-tree, we ask designers the context specific questions and apply a weighting scheme to valuate responses. For this questionnaire part, DPR knowledge-base will be used. At the end, patterns will be ranked according to the total weight and the pattern with the highest total will be selected. For secondary-level recommendation, in Figure 6, we select patterns to apply on a initial set of models. For the input, we have some initial designs of the system, in addition with inputs from that of primary-level. We convert those designs (e.g., UML models) to parsable XML documents. After parsing those XMLs, we perform keyword mining within the system for some abstract design elements, i.e., classes, methods, their relationships, etc. Finally, a systematic similarity checking will be performed, e.g., documents matching, for choosing a pattern for that particular sets of design with some pattern specifications (i.e., KB). We continue until all the XMLs already are under certain selected patterns.

C. Process Summary

Figure 5 summarizes the concepts of primary and secondary pattern recommendation, showing main input parameters, procedural steps and output for our DPR prototype in two different levels. For the first level of recommendation, DPR considers mainly user experience and knowledge on design patterns and the pattern knowledge-base introduced by our prototype. During the processing phase, first level recommendation relies on questionnaire using GQM model [2], use of a weighting scheme, and finally ranking for selecting a pattern in the output phase. For the second level recommendation, all the input elements from the first level are used as input factor, additionally some initial designs are also used that will be re-organized according to some pattern specifications.

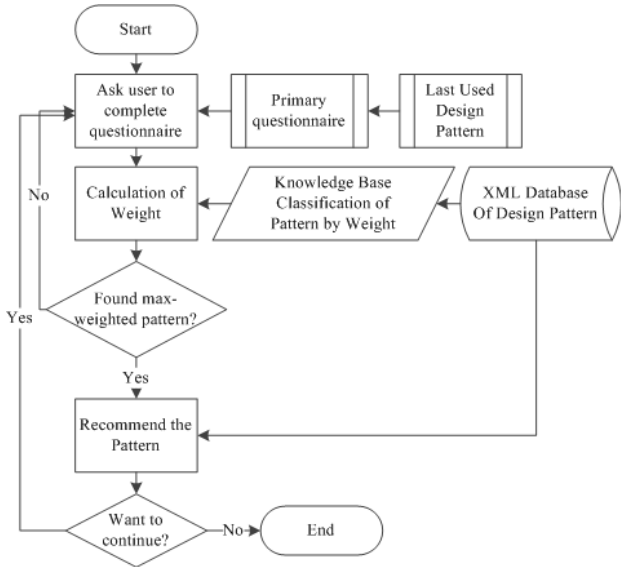

Figure 4. Process model: Primary recommendation

V. Weighting and Pattern Selection

We use a simple case study to evaluate our approach. Company A develops a program in Java to display all of its product details in the console. This Java program is simple. It takes an Iterator collection and iterates it to display product detail one by one. However, Company A outsource Product back-end system to a vendor called Vendor B. Vendor B came out with a system that will return all

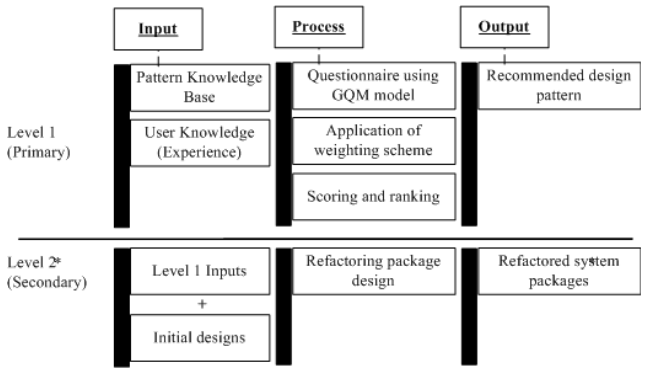

Figure 5. DPR Process Summary: primary & secondary

products as Enumeration collection. But returning Product as Enumeration will be a problem while Company A system is design to accept product as Iterator. Company A design are all based on Iterator collection to function, now Vendor B uses an old and obsolete type (Enumeration). They require a solution to display the products somehow. Above scenario sounds common, and here comes *Adapter*. What designer must create is an adapter class that can convert Enumeration to Iterator. We simulate this scenario for our case study.

To illustrate the pattern selection procedure, we present a high-level overview of the interaction between the system and the designer (see Table I). Based on the response of level-0 question, all the design patterns and their internally constructed GQM model will be presented to the designer as well-formed questions, which have either 'yes' or 'no' or 'do not know' responses and the corresponding weight. Let us say, designer responded with the 'Architectural' pattern for the level-0 question. To show an example, we present here only three candidate design (two architectural and one behavioral) patterns, i.e., Adapter, Decorator and Visitor. Visitor is related to Adapter in its applicability. The first question, 'Do you need to use several existing subclasses?' is dedicated to Adapter pattern. So, if designer responds with 'yes' and a weight of 8, then this value will be added to Adapter total. For all the 'yes' responses, the corresponding weight will be directly added to the corresponding patterns. Also, 'no' response will affect all the patterns other than Adapter and Visitor. For a question with response 'no', when there is no link, the weight will impact all the other patterns' total. For a question, i.e., question 10, if the response is 'Do not know', then by default the '0' will be added to the total indicating that it has no impact. For this example, in Table I, the Adapter pattern has the total weight of 59, Visitor 40 and Decorator of 13. So, DPR will suggest Adapter as it has the highest total weight (59).

VI. Evaluation

A. Subjective Evaluation

We conducted a small experiment with the DPR prototype for primary-level recommendation. Our experiment involved six graduate students and two IT professionals, hence referred to as subjects. We give a small introduction of the DPR core and the case study (in Section V). Subjects were asked to simulate the GQM model. At the

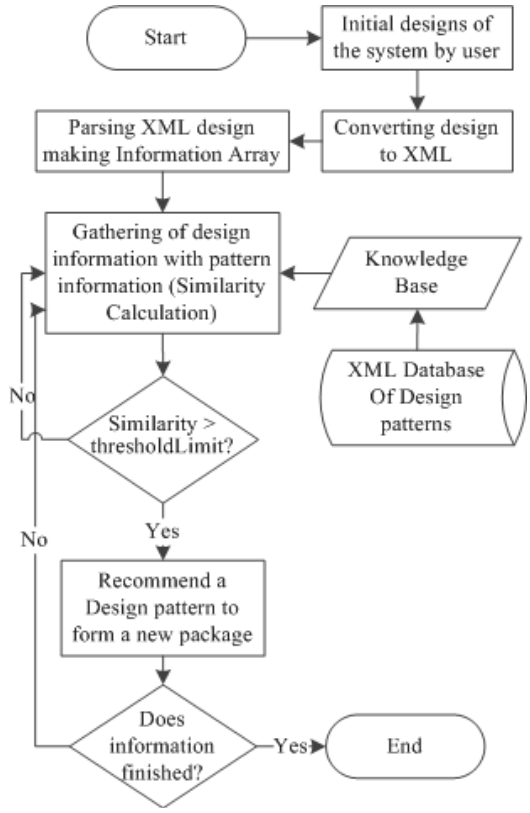

Figure 6. Process model: Secondary recommendation

Table I
A Designer-DPR Interactive Session: Response(R), Weight(W), Adapter(A), Visitor(V) and Decorator(D)

Questions	R	W	A	V	D
1. Do you need to use several existing subclasses?	yes	8	8	-	-
2. Do you want to create a reusable class?	yes	8	8	-	-
3. Do you want to use existing class?	yes	9	9	9	9
4. Do you want to perform operations on objects?	no	4	4	-	4
5. Do you want to avoid polluting classes with new operations?	yes	7	-	7	-
6. Do you want to define new operations over the structure?	no	6	6	-	-
7. Do you want to add responsibilities to individual objects dynamically without affecting other?	no	7	7	7	-
8. Do you want to use class for responsibilities that can be withdrawn?	no	9	9	9	-
9. Is the extension by subclassing is impractical for your problem?	no	8	8	8	-
10.Do you want your clients to be able to ignore different compositions of objects and individual objects?	-	0	0	0	0
11.Do you want to represent part-whole hierarchies of objects?	yes	7	-	-	-
Total weight given by the designer		71	59	40	13

Table II
SUMMARY OF THE SUBJECTIVE EVALUATION FOR DPR

Sub	OO	DP	NoQ	TotWeight	Pattern
1	medium	beginner	11	$\geq$51	Adapter
2	beginner	beginner	11	$\geq$51	Visitor
3	advanced	medium	10	$\leq$50	Adapter
4	medium	beginner	11	$\geq$51	both
5	advanced	medium	11	$\leq$50	Adapter
6	advanced	medium	11	$\leq$50	Visitor
7	medium	low	11	$\leq$50	Visitor
8	advanced	beginner	11	$\leq$50	Visitor
	Summary				
	beginner	low	NoQ$\leq$5	$\leq$50	succeed
	12.5%	12.5%	0%	62.5%	50
	medium	beginner	NoQ$\geq$6	$\geq$51	failed
	37.5%	50%	100%	37.5%	50
	advanced	medium			
	50%	37.5%			

Table III
COMPARISON: DPR, ESSDP, REBUILDER AND RS

Approach	Pros/Cons	Additional Features
DPR	Pros: max 11 Questions, selection based on ranking, flexible weighting scheme Cons: success certainity $\pm$50%	knowledge base, package refactoring, user session
ESSDP [8]	Pros: 60-88% certainity($\geq$ DPR), selection based on ranking Cons: 14-18 Questions($\geq$ DPR)	knowledge base, runtime reasoning
ReBuilder [9]	Pros: shows pattern list, ranked-list of results Cons: requires patterns pre-processing, no relevance score, patterns details not shown	human & machine readable pattern format
Recommender System [6]	Pros: Simple Cons: Suitable only for new designers, Unable to handle complex, does not use collaborative filtering, no feedback from users & large design problem	based on textual descriptions of DPs key-word based problem specification

end of the experiment, subjects were also provided with a questionnaire. We consider our experiment to be partial and preliminary due to small number of subjects and lack of detail assessment. However, this initial assessment is considered useful, as it helps us identifying weaknesses and improvements. For example, we may need to add more pattern specific or contextual questions that will facilitate users and ease selection pressure.

B. Discussion and Comparison

In Table II, OO column denotes how good subjects knew overall OO concepts. The level of knowledge on design pattern (DP column) was not very convincing, i.e., 50% of the subjects were at the beginner level. We had a set of total 11 questions. But, 50% of the subjects could identify the correct pattern and 50% suggested different patterns. For our case study in Section V, the correct pattern was 'Adapter'.

An interesting user behavior that we can observe from Table II, users with OO level as *advanced* or *medium* and DP level as *medium*, could mostly suggest the right pattern for the case study presented in Section V. That indicates the significant importance of user experience and knowledge, for applying and selecting design patterns during system design. Hence, the small subjective evaluation positively indicates the effectiveness of the proposed DPR prototype. Table III shows a comparison among different approaches in the literature [10]. Based on this survey and our preliminary results with DPR, from Table III, we can draw a clear conclusion that DPR is more efficient with a few additional features, compared to ESSDP [8], ReBuilder [9] and Recommender System [6] in terms of flexible weighting scheme and package (models) refactoring scope.

VII. CONCLUSIONS AND FUTURE WORK

We presented a DPR prototype (Section IV) for suggesting design patterns using an interactive session. We used an GQM approach [2]. Also, for secondary level recommendation, we give a high-level overview of the approach. Our knowledge-base (KB) contains all the pattern details and relative information (Section IV-A). In our DPR prototype, all 23 design patterns identified by Gamma et al. [4] can be included. We present a sample interactive session with designer in Table I. Our preliminary evaluation of DPR (Table II) by eight subjects shows DPR is relatively effective than the most closest literature work [8] by minimizing the number of questions and maximizing selection flexibility. As our current and future work, we are trying to come up with a tool that can suggest design patterns providing the interactivity for designers. More specifically, for secondary level recommendation, deployment of DPR for re-organizing model components into groups, for individual selected pattern, can be a prominent solution.

REFERENCES

[1] Sommerville and Sawyer. Requirements Engineering: A Good Practice Guide. 1997.

[2] N. E. Fenton, S. L. Pfleeger Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co, Boston, MA, USA, 1998.

[3] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. A Pattern Language. Oxford University Press, New York, 1977.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. 1995.

[5] J. K. et al. Foreword, Proc. 2007 ACM Conf. Recommender Systems. ACM Conference Recommender Systems, pp-3, 2007.

[6] Y. G. Guéhéneuc, and M. Rabih. A Simple Recommender System for Design Patterns, Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories (2007)

[7] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transaction Software Engineering, October 2005.

[8] R. S. David C. Kung, H. Bhambhani and G. Pancholi. An expert system for suggesting design patterns - a methodology and a prototype. In Software Engineering with Computational Intelligence, 2003.

[9] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, José Luis Ferreira and Carlos Bento 2002. Using CBR for Automation of Software Design Patterns. In Proceedings of the 6th European Conference on Advances in Case-Based Reasoning, pp:534-548.

[10] A. Birukou, A Survey of Existing Approaches for Pattern Search and Selection, EuroPLoP 2010, Bavaria, Germany.