Software Refactoring – an Approach Based on Patterns

Violeta Bozhikova*, Mariana Stoeva*, Bozhidar Georgiev*, Dimitrichka Nikolaeva* and Márta Seebauer**
*Department of Software and Internet Technologies, Technical University – Varna, Varna, Bulgaria
**Alba Regia Technical Faculty, Obuda University, Hungary

***Abstract*—The paper presents an approach for software development based on patterns. On one side, these are patterns in the role of best practices for software development on the other – patterns as bad solutions that must be avoided. Refactoring is a general way to transform a bad solution in a better one. This is a process of source code restructuring with the goal to improve its quality characteristics without changing its external behaviour. In refactoring we replace one software solution with another one that provides greater benefits: code maintainability and extensibility are improved, code complexity is reduced. The developed approach is implemented in software system that can be successfully used both in the real software engineering practice and in software engineering training process.**

***Key words*—Software Refactoring, Software Design Pattern Generation, AntiPattern Identification**

I. INTRODUCTION

The paper proposes an approach for software development based on both: Anti-Patterns detection and Design Patterns identification and generation. The presence of Anti-Patterns and Design Patterns is recognized as one of the effective ways to measure the quality of modern software systems. Patterns and AntiPatterns are related [1]. The history of software production shows that Patterns can become AntiPatterns. It depends on the context in which a Pattern is used: when the context become inappropriate or become out of date than the Pattern becomes AntiPattern. For example, procedural programming, which was Pattern at the beginning of software production activity, with advances in software technology gradually turned into AntiPattern. When a software solution becomes AntiPattern, methods are necessary for its evolution into a better one. Refactoring is a general way for software evolution to a better version. This is a process of source code restructuring with the goal to improve its quality characteristics without changing its external behaviour. In refactoring we replace one software solution with another one that provides greater benefits: code maintainability and extensibility are improved, code complexity is reduced ([2]÷[10]).

Based on the approach proposed a web system was developed. The system can be used as instrumental tool in the real practice of software production as well in the teaching process - to support several software engineering disciplines in "Software and Internet technologies" Department of the Technical University in Varna. The final effect of its application is to improve the software quality. It relies on techniques that generate the structure of Software Design Patterns, find AntiPatterns in the code and perform Code Refactoring.

Next section of this paper comments the structure and the basic components of the proposed approach. After, the software implementation of the approach is discussed; the system's architecture and the basic structural elements are presented.

II. APPROACH FOR SOFTWARE DEVELOPMENT BASED ON PATTERNS AND REFACTORING

The general structure of our approach is presented in figure 1. It takes as input the software source code that has to be refactored. The output is refactored code. The approach relies on accumulation of knowledge about the best practices in programming so "Accumulation of Knowledge" is one of the processes that are performed in parallel with other processes. The refactored code is result of "AntiPatterns Identification and fixing" and "Design Patterns Generation". Before generate Design Patterns it is necessary to analyze the code with the goal to find Design Patterns candidates. The proposed approach comprises the following main component:

A. Accumulation of Knowledge

Aims to provide information on design patterns and AntiPatterns. It contains information about creational, behavioral and structural design patterns and software AntiPatterns in software development and software architecture. Describe the problems that each design pattern solves, the advantages that it provides and the situations in which it is used. For the AntiPatterns - the nature of the problems and possible options for their avoidance are described.

B. Design Pattern Generation

Provides functionality to generate sample implementations of the design patterns structures; basic elements and relationships between them are generated, it's not implementation of the solution of specific problem. To generate a template user must select appropriate names for key elements. Appropriate names for the key elements must be given by the user, in order to generate a pattern.

C. Refactoring Component:

Provides methods for automatic code refactoring. The code is supplied as input of any method, as for inputs are accepted only properly constructed classes. Each method

performs the appropriate changes and returns the modified code as a result.

D. *AntiPatterns Identification and fixing*

Provides methods for code analysis. The code is supplied as input of a particular method and as result of code analysis the poorly constructed sections of code are colored. The colored code should be rewritten in order to increase its readability and maintenance.

E. *Design Pattern Identification:*

Provides methods to examine source code and to identify candidates for design patterns [6]. This component is still under development. Our detection strategy is based on the code inspection. Extensive research has to be conducted to develop techniques to automatically detect candidates of DP in the code.

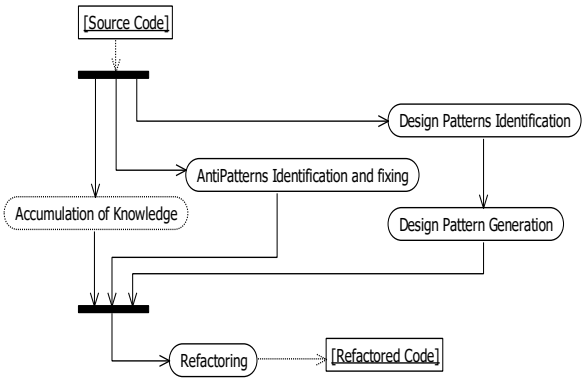

Figure 1. General structure of the approach for software development based on Patterns and Refactoring

III. SOFTWARE IMPLEMENTATION OF THE APPROACH

Based on the approach proposed a web system was developed. The basic structural elements of the web system are presented in figure 2.

A. *Main Page:*

It aims to present the different sections of the system with a short description and redirect the user to any of them.

B. *Encyclopedia:*

It aims to provide information on design patterns and AntiPatterns. It consists of two parts: menu type accordion and informative part. The user can select from the menu a concrete DP or AntiPattern. When you choose a concrete pattern then the information about it is displayed in the informative part. The section describes the problems that each design pattern solves, the advantages it provides and the situations in which it is used. For AntiPatterns – their nature and options to be avoided are described. This section is realized as one page with dynamic content that is changed through asynchronous AJAX requests to the server.

C. *Design Pattern Generation:*

This section offers functionality to generate sample implementations of the structure of the design patterns. The user chooses a type of pattern, inputs its parameters and click button "Generate". The generated code is displayed below the form. An example of design pattern (Template Method) generation is presented in figure 3.

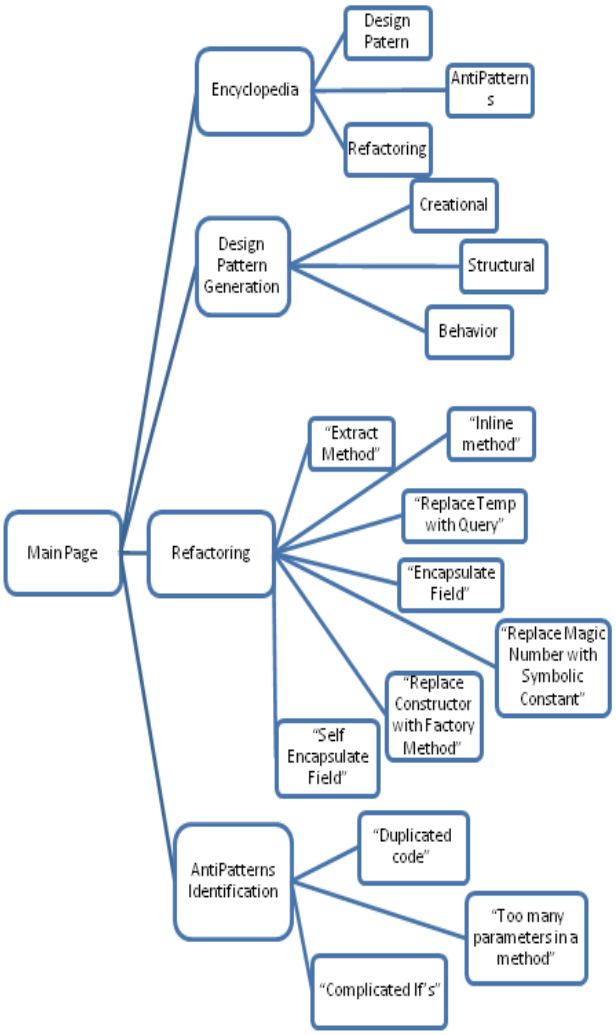

Figure 2. Basic structural elements of the web system

D. *Refactoring:*

This section provides 8 methods for automatic code refactoring: “Extract Method”, “Inline method”, “Replace Temp with Query”, “Encapsulate Field”, “Replace Magic Number with Symbolic Constant”, “Replace Constructor with Factory Method” and “Self Encapsulate Field”. Each method performs the appropriate changes of the code and the modified code is returned as a result. 8 refactoring methods are provided by the tool. In the left section of the refactoring window (figure 4), the user puts the code, which must undergo refactoring. After entering the necessary parameters the user has to press button "Refactor". The refactoring code is displayed in the right pane.

E. *AntiPattern Identification:*

This section offers methods for code analysis with the goal to detect AntiPatterns (“Duplicated code”, “Too many parameters in a method”, “Complicated If’s”). The code is supplied as input to each method, which analyzes

and paints the poorly constructed code sections. Then poorly constructed pieces of code must be rewritten to improve code readability and maintenance. Selecting a method (for example “Duplicate code”) a page for entering the code for analysis is visualized. After entering the necessary input data the user must press the button "Identify". The program will process the code and will paint the problematic code parts in red (the result is shown in the right section – figure 5).

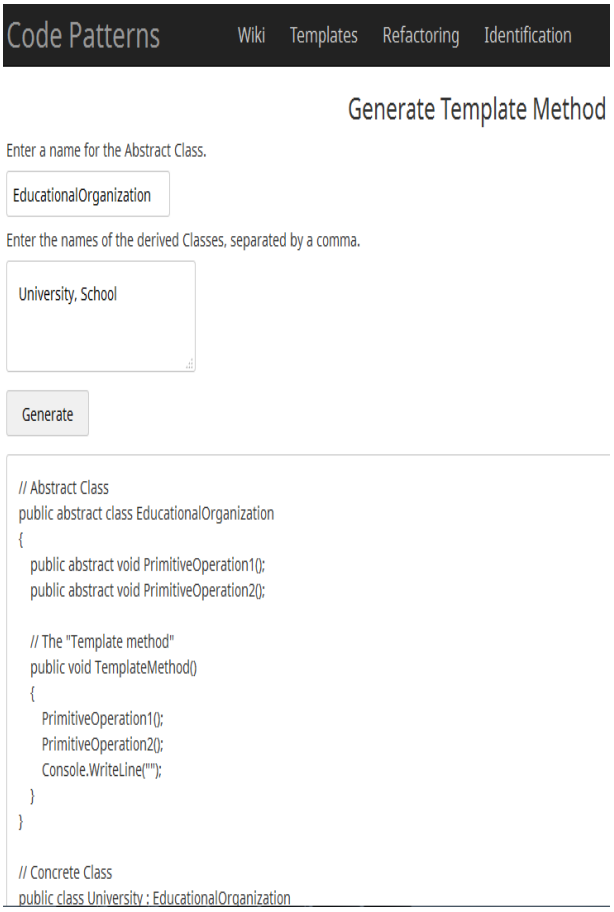

Figure 3. Template Method generation

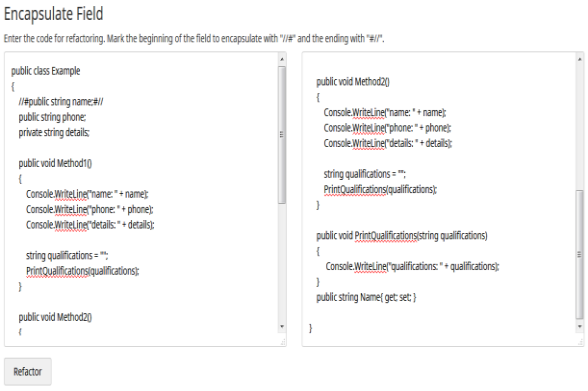

Figure 4. Refactoring window - method “Encapsulate Field”

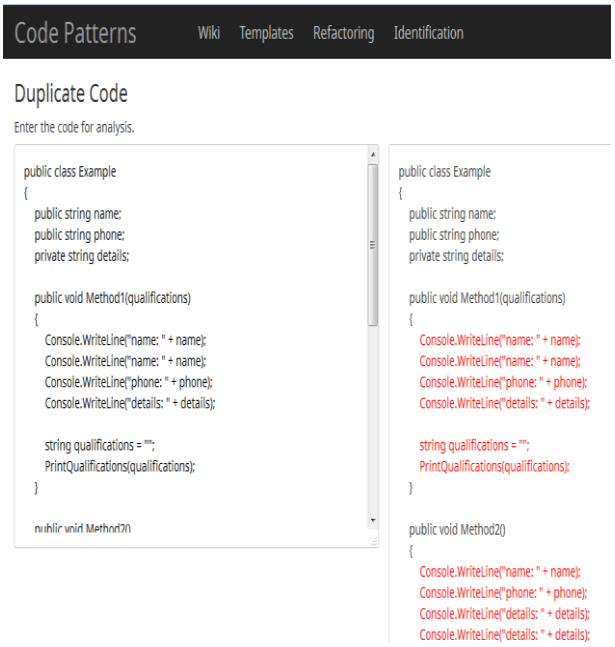

Figure 5. AntiPattern Identification window – “Duplicate code” identification

IV. CONCLUSIONS AND FUTURE WORK

The practical application of the developed software model in the practical exercises on the course “Computer Organization” has led to the following conclusions:

An approach for software development based on AntiPatterns detection and Design Patterns identification and generation is proposed in this paper. It relies on techniques that generate the structure of Software Design Patterns, find AntiPatterns in the code and perform Code Refactoring. Refactoring increases the software quality, it is a general way for software evolution to a better version.

The developed approach is implemented in a software system that that already has been applied in software engineering teaching process but could be also used in the real software engineering practice. It relies on the realization of 4 main sections: educational section that gives information on design patterns and AntiPatterns; Design Pattern generation section that offers functionality to generate the structure of 26 design patterns (Creational DP, Structural DP and Behaviour DP) in C#; AntiPattern identification section that at this time provides realization only of 3 methods for AntiPatterns detection; Refactoring section that provides 8 methods for automatic code refactoring. Each method performs the appropriate changes of the code and the modified code is returned as a result.

Our work associated with the approach presented and the system developed is still in its initial phase. We plan to add new patterns and AntiPatterns in encyclopaedic part. Support for languages other than C# can be provided by the Design Pattern generation Component. Future work is needed to implements more refactoring, AntiPattern and design pattern generation methods. In this time “Design Pattern Identification” section is only sketched and has not been studied and fully developed. So, we plan quite extensive research for the future implementation of this section.

REFERENCES

[1] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, Thomas J. Mowbray, AntiPatterns. Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, Inc., 1998, Canada

[2] https://en.wikipedia.org/wiki/Systems_development_life_cycle

[3] https://en.wikipedia.org/wiki/Code_refactoring

[4] Ali Ouni, Marouane Kessentini, Houari Sahraoui, Mel Ó Cinnéide, Kalyanmoy Deb, Katsuro Inoue, A Multi-Objective Refactoring Approach to Introduce Design Patterns and Fix Anti-Patterns, http://sel.ist.osaka-u.ac.jp/lab-db/betuzuri/archive/990/990.pdf

[5] Martin Drozdz, Derrick G Kourie, Bruce W Watson, Andrew Boake, Refactoring Tools and Complementary Techniques, https://pdfs.semanticscholar.org/ae2a/5ccaf697880cb386046e8882a6c268e83312.pdf

[6] Jagdish Bansiya, Automating Design-Pattern Identification, Dr. Dobb's Journal, 1998, http://www.drdobbs.com/architecture-and-design/automating-design-pattern-identification/184410578

[7] S. Negara, N. Chen, M. Vakilian, R. E. Johnson, and D. Dig, A comparative study of manual and automated refactorings, in 27th European Conference on Object-Oriented Programming (ECOOP), 2013, pp. 552–576

[8] M. Kim, T. Zimmermann, and N. Nagappan, A field study of refactoring challenges and benefits, in 20th International Symposium on the Foundations of Software Engineering (FSE), 2012, pp. 50:1–50:11

[9] Xi Ge and Emerson Murphy-Hill. Manual Refactoring Changes with Automated Refactoring Validation. In Proceedings of the Int. Conf. on Soft. Eng. (ICSE), 2014

[10] Ioana Verebi, A Model-Based Approach to Software Refactoring, https://www.researchgate.net/publication/281686403_A_Model-Based_Approach_to_Software_Refactoring