DEVELOPMENT OF A REFACTORING LEARNING ENVIRONMENT

A. Stoyanova-Doycheva

Abstract: *This paper describes a Refactoring Learning Environment, which is intended to analyze and assess programming code, based on refactoring rules. The Refactoring Learning Environment architecture includes an intelligent assistant – Refactoring Agent, which is responsible for analysis and assessment of the code, written by students in real time by using a set of refactoring methods. According to the situation and based on the refactoring method, which should be applied, the agent could react in different ways. Its goal is to show the student, as much as possible, the weak places of his programming code and the possible ways to makes it better.*

Keywords: Intelligent Agents, Refactoring, Tools, eLearning
2010 Mathematics Subject Classification: 97R40

1. Introduction

"Software Engineering: Computer Science Education and Research" [13] is an international project, funded by DAAD (German Academic Exchange Service) and realized under the auspices of "Stability Pact for South Eastern Europe". Thirteen institutions participate in it, one of which is the University of Plovdiv, Bulgaria. The coordinator of the project is the Institute of Informatics, Humboldt University, Berlin.

One of the main aims of the project was the cooperative research and practical experience gathering in reengineering of the currently active system XCTL in the field experimental Physics [19], where the task of the Bulgarian team was to realize the refactoring of the system.

During the work on this project we considered for the first time the concept of developing the Refactoring Learning Environment (rLE). On the basis of our results [15, 16] and particular theoretical models in addition, we decided to implement a programming tool, which we intend, on a further stage, to integrate in SELBO [14]. SELBO is a virtual environment, which we are currently working on. Its main aim is to assist both teachers and students in the field Software Engineering.

In this paper we present the programming tool Refactoring Learning Environment. It is intended to analyze and assess the code, written by students in real time, as well as to recommend to them changes in its structure, if needed, in order to improve its quality. The analysis and assessment are made by an intelligent assistant – Refactoring Agent (RA) in compliance with the rules for refactoring for the programming language Java, defined in [4].

2. Refactoring tools overview

In this section some of the available types of refactoring tools are briefly presented. Although refactoring process could be realized by hand, the possibility of applying automatic tools is of great importance. At present a number of such tools are available, where the aspect and degree of automation of the process vary depending on the particular tool and the maintenance it supports.

Tools such as Refactoring Browser [10], XRefactory [18], jFactor [5] apply semi-automatic approach after which the place and type of refactoring are chosen by the user.

Completely automatic refactoring, according to some scientific researches, is also an acceptable approach. Guru, for example, belongs to this category and is used for restructuring hierarchies of successors and methods for refactoring in SELF programs [9]. Some other approaches for automatic refactoring are presented in [1, 6, 11, 2].

A current tendency in this field consists in the integration of refactoring tools in powerful, industrial environments for development of software. Such is the case with Smalltalk Visual Works from v7, Eclipse from v2, Together Control Center from v6, IntelliJ IDEA from v3, Borland JBuilder from v7 etc. All these tools focus on applying refactoring in compliance with the user requirements.

Another group of tools, which are less in number in comparison with the previous ones, afford the opportunity to define when and where to apply refactoring. In [12] an approach is presented after which the implementation is realized via metrics, whereas in [8] the possibility of automation via invariants by means of the tool Daikon is described. The latter approach is based on a dynamic analysis of the behavior of the run-time of the system and its most proper application is as a complement to the other approaches.

3. rLE architecture

The rLE Architecture consists of two subsystems (Fig.1.):

- Front-end subsystem (FES) – the environment, which is used by the students for the development, compilation and testing of the source code;
- Back-end subsystem (BES) – the Refactoring Agent (RA), which is an intelligent agent that assists the students during the code development.

The Refactoring Agent is an autonomous software application that analyzes and assesses continuously the code that is developed in FES. Consequently, from the RA point of view FES is its environment.

The Refactoring Agent communicates with its environment by means of its sensors and effectors. Via the sensors RA accesses the complete source code. This implies not only the files, being edited, but also the completed ones that were not opened in FES for editing. This way the agent could make a deep analysis and give an adequate assessment for the required changes on the basis of all the code, and

not only of the part that is currently modified. The sensors provide also some basic metric information to the agent, which is used for initial filtering of the possible refactoring methods that can be further evaluated.

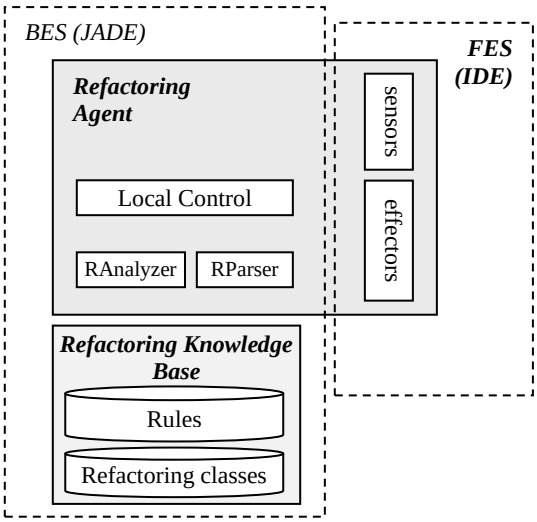

Figure 1. rLE Architecture

The possible metrics are LOC per class/method, number of methods/attributes per class and s.o.

The role of the effectors is to raise different events that assist the students by the accomplishment of their tasks in FES, where they are working. Such events could be:

- Underlying particular parts of the code by marking them with an appropriate color;
- Displaying messages in dialog windows, balloon messages etc.;
- Emitting sound-signals, vocal messages;
- "Materializing" the agent in the form of animation to exalt the effect.

The collaboration of the sensors and effectors is coordinated by the Local Control of the agent which is based both on the information, incoming from the sensors and the refactoring rules, stored in the Refactoring Knowledge Base (RKB) of the agent.

The analyzing of the source code, written by the student in FES, is made by the RAnalizer. Before RAnalizer starts his work, the RParser parses the source code and creates a tree structure from it. This tree structure can be analyzed by the RAnalizer.

The RKB consist of set of rules together with set of classes, which builds consistent knowledge base. Each rule describes in a common form the conditions, which allows a particular refactoring method to be put in the "short list", based upon some metrics.

By example a possible rule for choosing the "Extract class" refactoring method could be LOC_by_class > predefined_value.

In this way the rules are used by the RAnalyzer in order to make the initial filtering of the refactoring methods, which should be evaluated at the next step.

Each refactoring class contains the code for the particular refactoring method as well as code for final evaluation of the possibility to apply this refactoring

method. The refactoring methods filtered by the RAnalyzer are then examined by using the evaluation part of each refactoring class. In this way the agent takes final decision, which refactoring method at what place to be used.

As the last step the refactoring is applied by using the actual refactoring class after negotiation with the user – as described in the next topic.

The proposed environment differs from the existing decisions in several aspects:

- The environment is a prototype and is intended, first of all, for teaching students;
- The code analysis is done in real time, i.e. already during the development of the code the students could be assisted in improving its quality;
- An agent-oriented implementation is realized.

4. Agent functionality

Depending on the refactoring method, which should be applied, the agent could react in three different ways :

- To apply automatically the method after receiving confirmation from the user;
- To display detailed instructions, explaining to the user where and how the particular refactoring method should be applied;
- To ask the user additional questions in order to clarify the conditions and define the appropriate refactoring method.

Automatic Refactoring

In the cases when the refactoring method is comparatively simple and the criteria for its application are clear the agent could offer to the user to realize the required changes automatically. Some of the appropriate for these approach methods are: Move Method, Move Field, Extract Class, Extract Method etc.

Refactoring Proposal

Often the criteria for refactoring are clear but the application of the particular method implies a significant change in the code or its structure.

In these cases an approach is recommended after which the agent inform the user about the specific situation and offer him detailed explanations about the possible improvements that could be made in the particular situation.

Some of the proper refactoring methods that belong to this category are Replace Conditional with Polymorphism, Replace Delegation with Inheritance, Replace Inheritance with Delegation etc.

Refactoring Questionarie

Often the choice of applying one or another method for refactoring is made on the basis of an almost one-type set of criteria where just a few differ from one another.

In the cases when some of the requirements for applying the refactoring methods are met and yet this is not sufficient to define synonymously the most appropriate one, the agent could "ask" the user several questions in order to clarify the concrete situation.

After having made the requirements clear the agent defines the type of the situation again. It could be brought to one of the above described types: automatic refactoring or refactoring proposal.

5. Implementation

Taking into account the ever-increasing requirements towards the present-day Integrated Development Environments (IDEs) and the availability of open-source projects, which meet to a great extent these requirements, we chose Eclipse [3] as a development environment.

In addition Eclipse supports a powerful mechanism for interaction with external components in the form of plugins. This could be considered as a significant advantage of this particular environment which simplifies the integration of the Refactoring Agent (RA) in the development environment.

The sensors and effectors of the agent are realized as plugins in the IDE module. The agent itself is implemented by means of the JADE environment [7].

Current Implementation

The Refactoring Agent represents several classes written in Java that are embedded into the Eclipse Platform in the form of a plug-in. In this way the Refactoring Agent is able to access a particular java project's source code and additional infrastructure of the Platform i.e. graphical components and APIs.

The Eclipse platform consists of a core whose job is to run and manage hundreds and in some cases thousands of plug-ins. The plug-ins can and do use each other's APIs. The Eclipse UI Plug-in, for example, provides API for adding buttons, menus, etc. to Eclipse's graphical user interface. Many plug-ins use this API, and so does the plug-in in which the Refactoring Agent runs.

So, in order to integrate anything into Eclipse, we need to write a plug-in. That is exactly what we have done. In order to have a JADE agent analyzing and changing the code in the Eclipse Java Editor (which is part of the Java Development Tools plug-in), we need to start a JADE container and put an agent into it. Behaviors are then added to the agent. Behaviors hold the analyzing and changing logic, have access to and use the other plug-ins' APIs in order to do their job. The Refactoring Agent's behaviors also have access to the JADE API which can be used to communicate with other agents in the same or different containers and to delegate tasks to them.

After copying the jar file which contains the Refactoring Agent into Eclipse's "plugins" directory and launching Eclipse, a toggle button appears in on the toolbar.

When clicked for the first time after launching Eclipse, this toggle button creates and initializes a JADE container and the Refactoring Agent itself, which is a JADE agent and resides within the container. A repetitive behavior is then added

to the agent: every 5 seconds the agent's environment, namely the source code in the active Java editor, is scanned, and a syntax tree is generated. For example the syntax tree is searched for local variables that could be in-lined. Those variables, which are reassigned a value after initialization are not considered.

When found, the local variables that are possible to be in-lined are highlighted in the editor by changing their background, so that the student working with the Java file could see it.

Also on the left vertical ruler, Refactoring Agent's icons appear for every line that contains either the declaration or a usage of a local variable suitable for in-lining. On the right vertical ruler appear markers that when clicked, scroll the editor to the corresponding line of code. When any of the icons on the left vertical ruler are clicked, the corresponding code is selected and a dialog with options appears. The first option is the one offered by the Refactoring agent.

When double-clicked, this option in-lines the local variable - the declaration is removed and its usages are replaced with its value:

The information used to perform this action like positions in the source code is obtained from the generated syntax tree.

If the toggle button on the Eclipse's toolbar is pressed again, the agent's behavior is suspended until it is pressed once again. The highlighting of the code stops and the icons and markers on the left and right vertical rulers disappear.

Implementation of the Knowledge Base

The knowledge base of the Refactoring Agent is a fundamental part of its architecture. It contains the rule for determining a situation for the application of refactoring and the implementation of the refactoring methods.

Each class of the agent's knowledge base on refactoring includes a code, which realizes the specific refactoring method, and a code, by means of which is made a final evaluation of the possibilities for applying the refactoring method (the evaluation part). The refactoring methods, chosen by the RAnalizer, are under investigation. For that purpose there is used the evaluation part of the classes, which realize the refactoring methods in the knowledge base. In this way the agent takes a final decision about which refactoring method to use and in which location in the code, written by the student, to place it.

The current implementation of the RA knowledge base is presented on the next package diagram (figure 2).

The main package in the knowledge base is called "Pattern". It contains the common functionality for the rules and for the refactoring's methods. Each refactoring method is a set of classes that extends the abstract functionality from the "pattern" package in a way to reach the needed refactoring behavior. The refactoring's method set of classes are put in a different package for each refactoring method. The implementation of new refactoring method needs of two new classes that implement the concrete behavior.

Figure 2: Package diagram of the RA knowledge base

6. Conclusions

The Refactoring Learning Environment, proposed in this report, will be used in the Software Engineering Master's Program at Plovdiv University. The Refactoring Learning Environment enhances the creativity in the software engineering education. A crucial role in it is played by the Refactoring Agent, which is the cornerstone in the proposed architecture. Different reactions of the agent lead to different behavior of the students. Deciding with the Refactoring Agent's help which method of refactoring to use in the source code, the students can evince creativity. This makes the students' education in refactoring more efficient as it implements the "Learning-by-doing" strategy. The interaction between the refactoring agent and the student is a main part of the agent's activity, because this motivates the students to make decisions by themselves [17].

Up to now we have investigated 32 methods for refactoring from M. Fowler's book [4], which can be implemented in the current architecture of the RA. In future, the Refactoring Agent should be augmented with more logic for locating portions of source code suitable for refactoring and for providing options to resolve these situations

7. Acknowledgment

Asya Stoyanova-Doycheva wishes to acknowledge the support of the Bulgarian National Science Fund for Research Project Ref. No. ДО02-149/2008.

References

[1] Casais, E., „Automatic reorganization of object-oriented hierarchies: a case study", *Object Oriented Systems 1* (1994), pp. 95-115.

[2] Cinnèide, M., "Automated Application of Design Patterns: A Refactoring Approach," *Ph.D. thesis, Department of Computer Science*, Trinity College, University of Dublin (2000).

[3] Eclipse, URL *http://www.eclipse.org*

[4] Fowler, M., *Refactoring: Improving the Design of Existing Programs*, Addison-Wesley, 1999.

[5] Instantiations, jFactor (2002), URL *www.instantiations.com/jfactor/*

[6] Jahnke, J. H., A. Zündorf, "Rewriting poor design patterns by good design patterns", S. Demeyer and H. Gall, editors, *Proc. of ESEC/FSE '97 Workshop on Object-Oriented Reengineering*, Technical University of Vienna, 1997, Technical Report TUV-1841-97-10.

[7] Java Agent Development Framework, *http://jade.tilab.com/*

[8] Kataoka, Y., M. D. Ernst, W. G. Griswold, D. Notkin, "Automated support for program refactoring using invariants", *Proceedings of the International Conference on Software Maintenance* (2001), pp. 736-743.

[9] Moore, I., "Automatic inheritance hierarchy restructuring and method refactoring", *Proc. Int'l Conf. OOPSLA '96, ACM SIGPLAN Notices* (1996), pp. 235-250.

[10] Roberts, D., J. Brant, R. Johnson, "A refactoring tool for Smalltalk", *Theory and Practice of Object Systems* 3 (4) (1997), pp. 253-263.

[11] Schulz, B., T. Genssler, B. Mohr, W. Zimmer, "On the computer aided introduction of design pattern into object-oriented systems", *Technology of Object-Oriented Languages and Systems* (1998), pp. 258-267.

[12] Simon, F., F. Steinbrückner, C. Lewerentz, "Metrics based refactoring", *Proc. European Conf. Software Maintenance and Reengineering* (2001), pp. 30-38.

[13] Software Engineering: Computer Science Education and Research Cooperation, URL *http://www2.informatik.hu-berlin.de/swt/intkoop/daad/*

[14] Stoyanov, S., D. Mitev, I .Minov, T. Glushkova, eLearning Development Environment for Software Engineering Selbo 2, In: Proc. of the 19th International Conference on Database and Expert Systems Application (DEXA 2008), 1-5 September 2008, Turin, Italy, pp. 100-104, 2008, ISBN: 978-3-540-85653-5.

[15] Stoyanova-Doycheva A, B. Botev, R. Gospodinov, *Description of changes – Deffractometrie/Reflectometrie – use case AreaScan*, Third International Workshop "Software Engineering Education and Reverse Engineering", Ohrid, Macedonia, 25-30 August, 2003.

[16] Stoyanova-Doycheva A, R. Gospodinov, B. Botev, *Description of refactorings made on use case AreaScan, metrics, research about automated refactoring tools and future plans*, Third International Workshop "Software Engineering Education and Reverse Engineering", Ohrid, Macedonia, 25-30 August, 2003.

[17] Todorka Glushkova, Asya Stoyanova, *Interaction and adaptation to the specificity of the subject domains in the system for e-Learning and distance training DeLC, International Scientific Conference "Informatics in scientific knowledge"*, Varna Free University "Chernorizets Hrabar", 26-28 June, 2008

[18] XRef-Tech, XRefactory (2002).URL *xref-tech.com/speller/*

[19] XCTL, http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/intkoop/se/XCTL-Man-Adj.htm

Faculty of Mathematics and Informatics
University of Plovdiv
236 Bulgaria Blvd.
4003 Plovdiv, Bulgaria
e-mail: astoyanova@uni-plovdiv.bg