The "Extract Refactoring" Refactoring

Romain Robbes and Michele Lanza
Faculty of Informatics, University of Lugano - Switzerland

Abstract

There is a gap between refactoring tools and general-purpose program transformation tools that has yet to be filled. Refactoring tools are easy to use and well-established, but provide only a limited number of options. On the other hand, program transformation tools are powerful but are viable only for large transformation tasks. We propose an approach in which a developer specifies transformations to a program by example, using an IDE plugin recording the programmer's actions as changes. These changes could be generalized to specify a more abstract transformation, without the need of a dedicated syntax. Defining refactorings and transformations from concrete cases would enable more frequent uses of medium scale transformations.

1 Introduction

Refactoring [1], [2] has become a well-established program restructuring technique. Indeed, several major Integrated Development Environments feature a refactoring engine which automates the most common refactoring operations [3]. However, the refactorings supported by a refactoring engine are often limited in number and extent: only a fixed number of transformations are implemented. If a more complex change to a program is needed, it must either be done manually, or with the help of a generic program transformation tool.

Such program transformation tools [4], [5] are very powerful and allow large scale transformations to be performed with a much lower cost than if done manually. However, these tools still have a rather high barrier to entry, making them only suitable for large-scale transformations: They require the user to learn a transformation syntax and to have a high capacity in abstracting and reasoning at the meta level in order to define the transformation. [5] describes how a tool named DMS was used to migrate an application from one component style to another. They mention that such an approach is not wortwhile for small applications.

2 Restructuring a Program by Example

To fill the gap between refactorings and program transformations we propose an approach based on change recording and generalization. In such an approach a programmer provides concrete instances of a transformation manually and generalizes them to fully specify a transformation. This approach relies on a framework which records a programmer's actions in an IDE and model them as program transformations or *changes*. Each of these changes takes as input an abstract syntax tree (AST) of the system being monitored and returns a modified AST of the program.

Transformations recorded this way operate on specific entities of the AST (*e.g.* add method *setConcreteBar* to class *AbstractBar*). To define a generic transformation, a programmer takes this concrete transformation and progressively abstracts it until his goal is reached (*e.g.* add method *setConcrete***X** to class *abstract***X**).

These transformations would then be instantiated: The programmer would fix set the variables of the transformation (telling which class is **X**), before executing it. He would then evaluate the results and modify the transformation before retrying, should the result be incorrect.

3 Example

A programmer, Bob, discovers that class **Bar** from the system he is working on has too many responsibilities. **Bar** should be split in two classes: each instance of **Bar** should hold an instance of class **Baz**. The behavior encoded in **Baz** could thus vary if a subclass of **Baz** is given. This change is not trivial: Several methods in **Bar** need to move in **Baz**, and be replaced by delegation stubs. In addition, some accesses to instance variables of **Bar** need to be replaced by accessors.

To implement this change, Bob performs it first concretely, by delegating method **foo** from **Bar** to **Baz** and changing a direct access to variable **bag** to an accessor. Bob then examines his actions in his change history to generalize his change. He ends up with a generic change affecting two classes **A** and **B**, a variable **v** and a set of methods **SM**. The

transformation moves the implementation of the methods in **SM** from **A** to **B**, defines delegation stubs in **A** (forwarding the call to instance variable **v**, an instance of **Baz**), and replaces accesses of variables belonging to **Baz** by accessors.

Bob then applies the change to the method **foo** he first modified to verify that the results are the same. He then applies it to all necessary methods in class **Bar**, and can store the transformation should he need to move behavior across classes in the future.

4 Related Work

Apart from program transformation tools, our work is close to the field of programming by example [6], [7]. Programming by example consists in recording user actions and generalize them in a program. Our approach is based on the same principle, but is restricted to defining transformations.

Boshernitsan and Graham defined a visual language aimed at easing program transformations [8]. The transformation task is simplified, but programmers still have to specify the transformation: They can not provide a concrete instance of it.

5 Conclusion

We proposed an approach in which programmers can specify program transformations by giving concrete examples of them. Transformations should be expressed more easily and hence used more often than with current approaches.

The ideas described in this paper are partially implemented. Recording developer actions and converting them to change operations is provided by our prototype, Spyware [9]. Spyware has been previously used for software evolution analysis. Change generalization and application need to be implemented, and a suitable user interface should be built.

References

[1] Opdyke, W.F.: Refactoring Object-Oriented Frameworks. Ph.D. thesis, University of Illinois (1992)

[2] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison Wesley (1999)

[3] Roberts, D., Brant, J., Johnson, R.E., Opdyke, B.: An automated refactoring tool. In: Proceedings of ICAST '96, Chicago, IL. (1996)

[4] Roberts, D., Brant, J.: Tools for making impossible changes - experiences with a tool for transforming large smalltalk programs. IEE Proceedings - Software **152** (2004) 49–56

[5] Akers, R.L., Baxter, I.D., Mehlich, M., Ellis, B.J., Luecke, K.R.: Reengineering c++ component models via automatic program transformation. In: WCRE, IEEE Computer Society (2005) 13–22

[6] Halbert, D.C.: Programming by Example. Ph.D. thesis, Dept. of EE and CS, University of California, Berkeley CA (1984) Also OSD-T8402, XEROX Office Systems Division.

[7] Lieberman, H.: Your Wish Is My Command — Programming by Example. Morgan Kaufmann (2001)

[8] Boshernitsan, M., Graham, S.L.: Interactive transformation of java programs in eclipse. In: ICSE. (2006) 791–794

[9] Robbes, R., Lanza, M.: A change-based approach to software evolution. In: ENTCS volume 166, issue 1. (2007) 93–109

[10] Rubin, R.: Language constructs for pogramming by example. 3rd ACM-SIGOIS Conf on Office Information Systems, also SIGOIS Bulletin **7** (1986) 92–103