Refactoring

Martin Fowler
ThoughtWorks
Melrose, MA, USA
http://martinfowler.com
fowler@acm.org

ABSTRACT

This tutorial is an example driven introduction to refactoring: a disciplined approach to changing the design of an existing code base.

1. INTRODUCTION

A common phenomenon to software systems is that of software entropy, which suggests that over time the design integrity of software decays under the accumulated pressure of modifications, enhancements, and bug fixes.

Refactoring is a technique to stem and even reverse this process. It is a disciplined approach to altering an existing code base in order to improve its design without introducing new bugs. At the moment it's commonly used as a manual technique, but increasingly tools are available to automate many of the more common refactorings.

2. THE ESSENTIAL PROCESS

The essence of refactoring is to carry out modifications as a series of small steps. Each of these transformations is called a refactoring, and if done correctly will introduce no change to the behavior of the system, hence these refactorings are often described as semantics preserving or behavior preserving.

Examples of refactorings include:

Extract Method: taking a fragment of code inside a subroutine and turning it into its own routine.

Rename Method: changing the name of a method and altering all the callers of the method to use the new name.

Replace Conditional With Polymorphism: Taking conditional logic and moving it to subclasses in an exiting inheritance hierarchy.

If done correctly, any of these changes preserve behavior. Although each change is trivial, indeed too trivial to be worth doing, the cumulative effect of multiple refactorings can make a profound change to a software design.

Currently refactorings are most often performed manually, and since humans make mistakes, this means that behavior may change. Thus it's essential to have a strong suite of automated tests for the code that's being refactored and to run these tests frequently during refactoring – preferably the suite should be run after every refactoring.

Although manual refactoring is the most common form, most refactorings are automatable. Currently there is considerable activity, particularly in the Smalltalk and Java communities, to provide refactoring tools that automate many common refactorings. Commercial and open source tools are available.

3. FURTHER INFORMATION

My own book [1] is currently the only text on refactoring. It provides a catalog of around seventy refactorings together with tutorial information on how to do refactoring.

The refactoring home page [2] provides a barely inadequate portal for refactoring with links to refactoring tools, other references, a catalog of refactorings and details of a refactoring mailing list.

4. ACKNOWLEDGMENTS

My efforts to describe and teach refactoring depend heavily upon the works of other people. In particular I need to acknowledge Ward Cunningham and Kent Beck whose Smalltalk development style showed the importance of refactoring in the development process. I must thank Ralph Johnson and his group at the University of Illinois at Urbana-Champaign for their key academic contributions: in particular the doctoral works of William Opdyke, John Brant and Don Roberts. The latter two also deserve thanks for their ground-breaking refactoring tool for Smalltalk

I must also thank countless people who have contributed to refactoring through writing down their ideas and experiences. In particular I wish to thank those who develop refactoring tools who, in my view, are making a huge contribution to the future of software development.

5. REFERENCES

[1] Fowler, M. Refactoring: Improving the Design of Existing Code, Addison-Wesley, Reading MA, 1997

[2] Refactoring home page. http://www.refactoring.com

ICSE'02, May 19-25, 2002, Orlando, Florida, USA.