Refactorings for Fortran and High-Performance Computing

Jeffrey Overbey, Spiros Xanthos, Ralph Johnson and Brian Foote
University of Illinois at Urbana-Champaign
MC 258
201 North Goodwin
Urbana, IL 61801
{overbey2,xanthos2,johnson,foote}@cs.uiuc.edu

ABSTRACT

Not since the advent of the integrated development environment has a development tool had the impact on programmer productivity that refactoring tools have had for object-oriented developers. However, at the present time, such tools do not exist for high-performance languages such as C and Fortran; moreover, refactorings specific to high-performance and parallel computing have not yet been adequately examined. We observe that many common refactorings for object-oriented systems have clear analogs in procedural Fortran. The Fortran language itself and the introduction of object orientation in Fortran 2003 give rise to several additional refactorings. Moreover, we conjecture that many hand optimizations common in supercomputer programming can be automated by a refactoring engine but deferred until build time in order to preserve the maintainability of the original code base. Finally, we introduce Photran, an integrated development environment that will be used to implement these transformations, and discuss the impact of such a tool on legacy code reengineering.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*program editors*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring*

General Terms

Design, Languages.

Keywords

Refactoring. Fortran programming.

1. THE NEED FOR REFACTORING

The quality of any long-lived code base tends to degrade over time. Otherwise stable architectures tend to take on a certain malleable quality as programmers attempt to adapt systems to meet unforeseen new requirements. Moreover, many coding best practices—e.g., small methods and concise, descriptive names—fall to the wayside when deadlines and functionality are in jeopardy. Although these issues are negligible in isolation, their cumulative action is often an erosion of the system's architecture [2]. Such systems tend to have high entropy, exhibit code duplication and global information sharing [5]. Maintenance and expansion can become tedious, costly, and time-consuming work [1].

One solution to this gradual software decay is refactoring [6, 10]. Refactorings are source-level program transformations that preserve the observable behavior of a system while improving its source code. Often, refactorings aim to eliminate code duplication or poor design decisions. Common refactorings include renaming variables or functions to be more descriptive, breaking a large subroutine into several smaller ones, or substituting one algorithm for another. In object-oriented systems, common refactorings include replacing `case` statements with polymorphism, introducting Method Objects, and moving methods between classes. Martin Fowler's *Refactoring: Improving the Design of Existing Code* [6] gives a more extensive catalog of refactorings.

One particularly interesting quality of refactorings is that many of them are algorithmic in nature. In essence, renaming a function amounts to a textual change of its declaration and of all invocations. While this is certainly not an easy task (due to complications such as preprocessing and function overloading), it can, in fact, be automated.

The Eclipse Java Development Tool [9] has arguably brought automated refactorings to the widest audience, although it was not the first tool to implement them. Much of the initial work in automated refactorings was done by Ralph Johnson's research group at the University of Illinois. William Opdyke's Ph.D. thesis [10] is often cited as the pioneering work in this area. John Brant and Don Roberts' *Smalltalk Refactoring Browser* [12] was the first implementation, introducing automated refactorings to the Smalltalk community. It has since been integrated into VisualWorks Smalltalk. More recently, Alejandra Garrido has been working with preprocessing issues in C refactoring [7], [8], and Photran will introduce refactorings specific to Fortran and high-performance computing.

2. REFACTORING FORTRAN

The classical use of refactoring as a form of retroactive engineering is particularly applicable to Fortran. The amount of Fortran 77 code that remains in production is a clear indication that software maintenance issues apply equally to HPC. However, there are many refactorings that are specific to the Fortran language.

In her M.S. thesis [4], Vaishali De identifies many possible Fortran refactorings. Many of the standard refactorings for fields and methods (described in [6]) apply equally to variables and subroutines in Fortran. Extract Method (i.e., removing a section of code into its own subroutine), Decompose Conditional (replacing a complex boolean expression with a more descriptive function call), Rename Variable, and Reorder Procedure Arguments are several examples. Some refactorings typically applied to classes in an object-oriented system can be applied equally well to Fortran modules—e.g., Encapsulate Field (where accesses of and assignments to a variable are replaced with function calls) and Move Method (moving a subroutine between modules).

Fortran also brings several unique refactorings. One example is transforming code from fixed format to free format. This is just one part of migrating code from Fortran 77 to Fortran 90/95. Another example would be migrating parallel arrays to derived types.

Similarly, once Fortran 2003 compilers materialize, existing Fortran programs will need to be migrated to that as well. One aspect of this will be especially challenging: Fortran 2003 introduces object orientation to the language. The process of transforming a procedural code base into an object-oriented one is certainly not one that can be automated completely: A great deal of domain knowledge and a number of design decisions must be made in the process. However, we anticipate that many of the steps can be automated, such as migrating module procedures to type-bound procedures.

3. REFACTORINGS FOR HIGH-PERFORMANCE COMPUTING

In the high performance world a very important issue for the software is the optimization for certain architectures. Many of the refactorings described previously can be applied to a large number of programs. However, there is an entirely different class of refactorings that are unique to the domain of supercomputing: namely, performance refactorings.

It is well-known that, despite the best efforts of compiler vendors, code intended to run on a specific supercomputer must undergo many hand optimizations. Examples include manual unrolling of loops and optimizing data structures based on the machine's cache size. Applying these tweaks by hand is a tedious error-prone process. So, a tool that would be able to automate the process of applying these tweaks would be very useful.

However, while refactorings are typically used to improve the design and readability of code, many of these performance optimizations actually decrease readability. Loop unrolling is an excellent example: An unrolled loop is far more difficult to comprehend than one that has not been unrolled.

In these cases, rather than transforming the code in-place, we propose the notion of deferred transformations.[1] In the same way that a breakpoint can be set on a specific line, programmers could tag a section of code to indicate that a given transformation (e.g., "unroll this loop five times") should be applied immediately before the code is compiled. This would allow programmers to maintain the more readable version of their code while compiling a performance-optimized version.

[1]We use the term "transformations" rather than "refactorings" since the transformations are not made to the working code.

4. REENGINEERING AND REFACTORING

Demeyer *et al.* define reengineering as "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" [13]. When we talk about reengineering, then, we usually refer to a legacy software system that has to be altered to meet changing requirements or to support extensions and additions.

Legacy code is a critical part of many systems and the source of many problems. Software systems must constantly adapt to changes of the environment in which they operate. In most cases, legacy software has been developed in obsolete languages by using old-fashioned development practices. This complicates the upgrade of legacy parts. One option is to discard or replace legacy parts, but this is not always feasible because of the cost. In this case, the most viable solution is to reengineer these parts.

During the reengineering process, we want to improve the design of the system in order to make it more maintainable and upgradable. This involves the alteration of its structure, but at the same the preservation of its behavior. The main reason that programmers don't attempt to improve the design of a system is their fear of breaking its behavior in the process [14].

When we alter the structure of a system in order to improve its design without changing its behavior, we are, by definition, refactoring. A tool that provides automated refactorings can eliminate the most of the common mistakes that programmers make when they attempt to manually refactor a piece of software. (Proper testing is still essential, however.) This shows the importance of a tool like Photran when someone is working with legacy code.

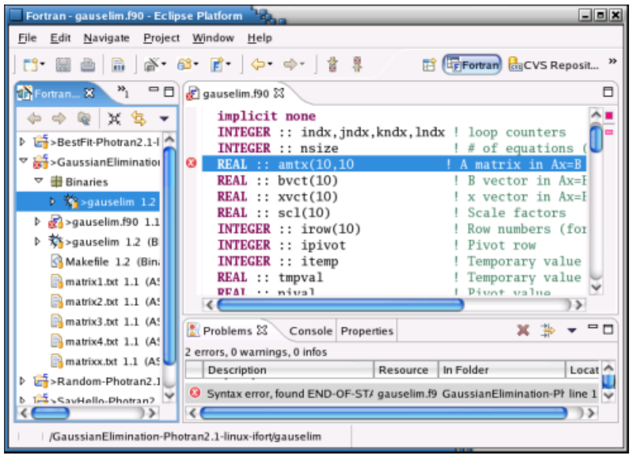
5. PHOTRAN

Identification of refactorings for Fortran and high-performance computing is an essential step, but even more important is the development of a tool that can automate them. Photran [11] is an Eclipse-based Fortran IDE being developed at the University of Illinois that will implement many of these transformations.

The current version of Photran is based on Eclipse's C Development Tool [3] and runs on Eclipse 3.0 under Linux, Windows, Solaris, and Mac OS X. It includes a keyword-highlighting Fortran editor, CVS support, debugging via a GUI interface to GNU `gbd`, Makefile-based compilation, and error extraction for several popular Fortran compilers.

A sophisticated refactoring infrastructure is under development, although it is not visible in the current public release. A pretty printer and Rename refactoring have been demonstrated internally and will be included in the near future. While we have identified many refactorings that will be essential for Fortran programmers, there are many we are not yet aware of. Receiving input from Fortran programmers, especially those in the high-performance arena, will be essential.

Our aim is to provide Fortran programmers with a state-of-the-art tool that can increase productivity and allow them to adapt their code to changing requirements and modern software engineering practices. Such a tool will be essential in reengineering efforts. Since Fortran has been used for more than fifty years, a vast amount of active Fortran code is decades old, making a tool like Photran a necessity for Fortran programmers.



```
implicit none
INTEGER :: indx,jndx,kndx,lndx ! loop counters
INTEGER :: nsize               ! # of equations (
REAL :: amtx(10,10             ! A matrix in Ax=B
REAL :: bvct(10)               ! B vector in Ax=B
REAL :: xvct(10)               ! x vector in Ax=B
REAL :: scl(10)                ! Scale factors
INTEGER :: irow(10)            ! Row numbers (for
INTEGER :: ipivot              ! Pivot row
INTEGER :: itemp               ! Temporary value
REAL :: tmpval                 ! Temporary value
REAL :: pival                  ! Pivot value
```

2 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Locat
Syntax error, found END-OF-ST/	gauselim.f9	GaussianElimination-Pł	line 1

/GaussianElimination-Photran2.1-linux-ifort/gauselim

Figure 1: A Screenshot of Photran 2.1

6. REFERENCES

[1] Boehm, B., and Horowitz E. (editors) *The High Cost of Software: Practical Strategies for Developing Large Software Systems.* Addison-Wesley, 1975.

[2] Brooks, F. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1995.

[3] *C Development Tool* http://www.eclipse.org/cdt/

[4] De, V. *A Foundation for Refactoring Fortran 90 in Eclipse.* M.S. Thesis, University of Illinois at Urbana-Champaign, 2004.

[5] Foote, B., Yoder J. *Big Ball of Mud.* Fourth Conference on Patterns Languages of Programs (PLoP'97/EuroPLoP'97). Monticello, Illinois, September 1997.

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

[7] Garrido, A., and Johnson, R. "Challenges of Refactoring C Programs." *Proceedings of IWPSE 2002: International Workshop on Principles of Software Evolution.* Orlando, Florida. May 19–20, 2002.

[8] Garrido, A., and Johnson, R. "Refactoring C with Conditional Compilation." *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003).* Montreal, Canada, October 6–10, 2003. 323–326.

[9] *Java Developement Tools for Eclipse* http://www.eclipse.org/jdt/

[10] Opdyke, W. *Refactoring Object-Oriented Frameworks.* Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1992.

[11] *Photran, an Eclipse plugin for Fortran Developement* http://www.photran.org/

[12] Roberts, D., Brant, J., and Johnson, R. "A Refactoring Tool for Smalltalk." *Theory and Practice of Object Systems* 3(4), 1997.

[13] Demeyer S., Ducasse S. and Nierstrasz O. *Object-Oriented Reengineering Patterns.* Morgan Kaufmann, 2003.

[14] Feathers M. *Working Effectively with Legacy Code.* Prentice Hall, 2004.