Software Reuse, A Fundamental Aspect of Software Engineering

Gertrude Levine
Fairleigh Dickinson University
1000 River Road
Teaneck, New Jersey 07666

levine@sun490.fdu.edu

Abstract

Structured programming, modular programming, information hiding, and abstraction are all widely recognized and accepted areas of good programming design. Software reuse, however, has just recently come under intense investigation as a software engineering discipline. Yet the economics of the construction of large software systems warrant the recognition and study of software reuse as a fundamental aspect of software engineering.

Keywords: Domain Analysis, Software Engineering, Software Reuse

1 Introduction

Fairleigh Dickinson University is offering, for the first time, a graduate course in Software Reuse in the Spring 1992 semester. Our computer science department recognizes the importance of software engineering, although individual faculty differ in the material to be covered.

A course entitled Software Engineering is a core requirement of the graduate program and a course called Software Techniques is an undergraduate requirements. Both of these courses include top down design, modularity, verification and testing, efficiency considerations, user interfaces, and documentation. In addition, the graduate course stresses the design of large systems, assigning large projects for team programming.

As part of my research interests, I have taught a special topics course using Ada several times, with a main focus on software engineering principles. Course work includes encapsulation, design of packages, generics, object oriented design, and specification, implementation, modification, and integration of library units.

I requested teaching a course in software reuse because many important issues are not covered or not adequately covered in any of our present course work. Yet these issues are fundamental to the control of the increased complexity of current and future software systems. Software factories that outsource and assemble software components may be the most cost effective method of constructing large systems; present technology in the United States does not support such "manufacturing." I am interested not only in the specification, construction, and cataloging of reusable software components, but in a methodology for putting such modules together and for testing these integrated systems. It is time to introduce into the study of software engineering, standards for reuse technology in general and the concepts of reuse architecture (i.e., domain specific software architecture or canonical architecture) in particular.

2 Course Introduction

The course is designed as a one semester graduate level seminar, with student research and experience contributing strongly to the material covered in class. Many of our graduate students hold responsible fulltime software positions; their strong working backgrounds have supplied supporting material for numerous lectures.

Thus the initial lecture will focus upon student experiences. All of our students have some reuse knowledge, although they may never have so identified it.

From the days when Grace Hopper shared cosine routines written in octal, software functions have been widely reused. Early applications were mathematical, with domains that were well understood. Such functions were quickly added to libraries that were included with compilers. More general subroutines were soon developed for broader purposes, but depending on the language, the domain of the arguments, and the documentation, they were not quite so easily understood or reused. How many students have written their own sort in C because they did not know how to utilize C's quicksort?

All students have had experience with sorts and searches; some have seen list components generalized with Ada's generic capability. Those students who have had exposure to Ada or object oriented languages can discuss the advantages of encapsulation for reuse. Communication through global data and through reference parameters are other familiar concepts that can be discussed in a reuse framework. Such examples illustrate, in a small way, the importance of programming environments.

Students probably all remember code written for one project that proved, perhaps partially, appropriate for another. Some may have written code specifically with reuse in mind. What did they place in the code to encourage reuse? What kind of documentation did they add? Did they include the reason for any distinctive features [Bail 91] of the design?

How often have students rewritten code that they could not find, although they "knew they had it somewhere?" Do they store and classify their work? How many times after they have written code have they found that someone else had solved the same problem and in a better way?

Some of our Ada students use a language sensitive editor to generate code. MIS students use CASE tools for their projects. They realize the role of tools in the generation of systems.

Many of our students have worked in the same application area for many years and have reused their system design for multiple customers. The student that designs office systems for doctors, for example, will probably testify that the first system was completed at a loss; only the ability to reuse it and customize it made the venture economically feasible.

All computer science students have some reuse experience. These will be collected and classified.

3 Course Material

After this introduction, we will be ready to formalize the course work. The textbook for the course is Software Reusability, Biggerstaff and Perlis, Volume I [Bigg 89]. Papers in Volume II will be used in discussion of topics in Volume I. In addition, other papers will be made available to the class. Each student will be required to be the expert on a different paper that will supplement class discussion on the papers of the textbook.

Discussion of the first three papers of the book [Bigg 87, Horo 84, Wegn 89] will be integrated. I will provide an outline for discussion, but feel that all three of these papers are important and complement each other in presentation of most of the issues.

The book is organized into sections of composition-based systems and generation-based systems, although there is overlap in material. Standards are sought for unit and system composition that achieve "pluggable" software components.

Programming design techniques [Parn 83], component classification and search techniques [Prie 87] and programming environments [Meye 87] are important and comparatively easy to understand. I plan to assign only two of the papers [Gogu 89, Litv 84] that emphasize theory for class discussion, although any of the others will be available for student presentation.

A particularly well written paper on generation-based systems [Neig 89] will have its discussion on domain analysis directly preceded by others papers [Deut 89, Jett 89, Rice 89]. Several weeks will be devoted to management issues such as quantitative results [Dure 91, Selb 89], since a reuse course should be effective for FDU's MIS Master's program as well. I have not yet chosen all of the papers that I consider important to the course, although the textbook contains enough for a strong basis.

4 Expected Results

A seminar course in software reuse must address a wide range of current issues. The papers chosen for discussion introduce most of these issues. The fact that so many overlap in material will aid team discussion.

As a secondary goal, students might repeat presentations for faculty. I am hoping to influence the department to update the material in the required software engineering courses, to include some of the basic concepts of software reuse.

Software reuse is perhaps the most cost effective methodology available for the development of software. Reuse architecture shows promise in the handling of the complexity of large systems. If the United States is to remain competitive in this field, faculty and students must become familiar with the material.

References

[Bail 91] Bailin, S. Correspondence.

[Bigg 89] Biggerstaff, T. and Perlis A., ed. Software Reusability. Concepts and Models. Vol. I & II. Addison-Wesley Publishing Co. Reading, Mass. 1989.

[Bigg 87] Biggerstaff, T. and Richter, C. “Reusability Framework, Assessment, and Directions”, *IEEE Software*, Vol. 4, No. 2, March, 1987 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Deut 89] Deutsch, L.P. “Design Reuse and Frameworks in the Smalltalk-80 System” in Biggerstaff and Perlis, Vol. II.

[Dure 91] Durek, T. “Software Production Through Reuse: Key to Improved Quality and Productivity.” Software Productivity Consortium, 1991.

[Gogu 89] Goguen, J. “Principles of Parameterized Programming” in Biggerstaff and Perlis, Vol. I.

[Horo 84] Horowitz, E. and Munson, J. “An Expansive View of Reusable Software”, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September, 1984 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Jett 89] Jette, C. and Smith, R. “Examples of Reusability in an Object-Oriented Programming Environment” in Biggerstaff and Perlis, II.

[Litv 84] Litvintchouk, S. and Matsumoto, A. “Design of Ada Systems Yielding Reusable Components: an Approach Using Structured Algebraic Specification”, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September, 1984 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Meye 87] Meyer, B. “Reusability: The Case for Object-Oriented Design”, *IEEE Software*, Vol. 4, No. 2, March, 1987 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Neig 89] Neighbors, J. ”Draco: A Method for Engineering Reusable Software Systems” in Biggerstaff and Perlis, I.

[Parn 83] Parnas, D., Clements, P. and Weiss, D. “Enhancing Reusability with Information Hiding”, *ITT Proceedings of the Workshop on Reusability in Programming*, Newport, R.I., 1983 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Prie 87] Prieto-Diaz, R. “Classification of Reusable Modules”, *IEEE Software*, Vol. 4, No. 1, 1987 (Reprinted in Biggerstaff and Perlis, Vol. I).

[Rice 89] Rice, J. and Schwetman, H. “Interface Issues in a Software Parts Technology” in Biggerstaff and Perlis, I.

[Selb 89] Selby, R. “Quantitative Studies of Software Reuse” in Biggerstaff and Perlis, II.

[Trac 88] Tracz, W. Software Reuse: Emerging Technology. IEEE Computer Society, Washington, D.C., 1988.

[Wegn 89] Wegner, P. “Capital-Intensive Software Technology”, *IEEE Software*, Vol. 1, No. 3, July, 1984 (Reprinted in Biggerstaff and Perlis, Vol. I).

5 Biography

Gertrude Levine is an associate professor of computer science at Fairleigh Dickinson University. Her present research interests are in conflict control, the Ada programming language, and software reuse. She has introduced into the FDU computer science curriculum undergraduate courses in computer networks and the Ada programming language, as well as a special topics graduate course in Ada, made possible through equipment grants from the New Jersey Technological/Engineering Education Grant Program. She has been contributing a column on Reusable Software Components to Ada Letters for the past two years.