Towards A Method of Design for Reuse

Bill Frakes

Software Productivity Consortium
2214 Rock Hill Rd.
Herndon, VA 22070

Abstract

An approach to derive reuse design principles is described.

Keywords: reuse design, reusable assets

1 Introduction

One trend in reuse has been a shift in interest from library issues to issues involved in the acquisition of reusable assets. I recently surveyed engineers and managers from some of the SPC member companies about which reuse issues they felt were most crucial to their companies. The results showed that factors related to the creation of reusable assets–design, domain analysis, and re-engineering–were perceived to be most important[2].

Based on this information, and my perception of problems in the reuse area, I am beginning a project to identify and test principles for the design of reusable components. The approach is to identify and organize such principles–the more abstract being applicable to more objects. The scope of the study will be all lifecycle objects. The study is based on the assumptions that (1) reuse design principles exist for many different languages and environments, (2) that commonalities exist between the principles in those environments, and (3) that more general principles can be abstracted from these lower level common principles.

2 Reusable Assets

There are three ways to get reusable assets; you can buy them, you can build them, or you can re-engineer them. All of these are dependent on an understanding of reuse design. To build a reusable part, one must know the characteristics that will make the part reusable. To re-engineer a

part for reuse, the part must be changed to have those characteristics. To buy a reusable part, we must know what characteristics to look for.

All three are dependent on an understanding of the domain involved. Domain analysis is the process of understanding a domain and writing down information about it in a somewhat formal way. Certain reuse design decisions will depend on the domain involved.

Reuse design principles will also be dependent on the type of the reusable object. Test cases, C functions, and Ada packages are all potentially reusable objects, but some of the methods for making them reusable will obviously be specific to their type. Object granularity, which is also a design discriminator, is subsumed by type.

Another factor that will affect reuse design is the environment the parts are used in. The way the parts are designed for reuse will be based on assumptions about the environment that they'll be used in. A good example of this is the Unix shell. In designing a shell tool one relies on the standardized byte stream interface. If this interface didn't exist, the part would have to be designed differently.

Thus, one might use the following example faceted scheme to classify reuse design principles.

Domain	Type	Environment
database	function	Unix
statistics	package	C
	test case	Fortran
	requirement	

Reuse is one of many design goals and must be traded off with some others. Reusability is usually inversely related to such design goals as efficiency of size and execution. Both Doug Lea and Mike Vilot have told me that simultaneously achieving reusability and execution efficiency in their libraries is necessary and difficult, but doable.

Several researchers [5], [4], [1], [6], have formulated a general model of design for reuse – the 3C's (concept, content, context) model. This model provides good insights into the reuse design problem, but does not provide detailed design guidance for specific LCO's. I'm not clear on how much this model overlaps with the scheme I'm proposing. I would like to discuss this point at the workshop.

3 Proposed Study

Phase 1 of the study will be to identify and classify reuse design principles into common groups. The classification will probably be hierarchical with the higher levels containing more general design principles and those at the lower levels containing principles more specific to given objects. A principle at the highest level, for example, might be:

"Identify and isolate the changeable parts of the reusable item"

A lower level principle, specific to a code function in a procedural language might be:

"Genericize the function by passing a pointer to the match routine."

The result of the first phase of the study will be a survey of existing reuse techniques and a preliminary classification of those techniques.

Phase 2 will be a test of the classification, and principles, on a set of reusable items. The output will be a report describing the results of the test. This phase would iterate until the investigators were confident of the results

Phase 3 would be the writing of a set of guidelines for reusable design. The information from this project could be used to augment various reuse products, such as style guides. The information that would be gained would also provide attributes of reusable components that would complement the reuse certification work that John Knight[3] is doing.

References

[1] Edwards, S. "An Approach for Constructing Reusable Software Components in Ada." Technical Report P-2378, Institute for Defense Analyses, September 1990.

[2] Frakes, W. "Software Reuse: Payoff and Transfer", Proceedings of AIAA Computing in Aerospace 8, October, Baltimore, 1991, pp. 829-831.

[3] Knight, J. "Reuse as a technique for achieving high reliability", Proceedings of the third annual workshop: methods and tools for reuse. New York State Center for Advanced Technology in Computer Applications and Software Engineering, Syracuse University 1990, CASE center technical report no. 9014.

[4] Latour, L. "A methodology for the design of reuse engineered Ada components" Ada Letters vol.11, no.3 p.103-13, Spring 1991.

[5] Tracz-W. "Modularization: approaches to reuse in Ada", J. Pascal Ada Modula-2 (USA). vol 9, no 5, pp. 11-12, 14-25, Sept.-Oct. 1990.

[6] Weide, B.W., Ogden, W.F., and Zweben, S.H., "Reusable Software Components," chapter in Advances in Computers, M.C. Yovits, ed., 1991, to appear.

4 Biography

Bill Frakes is Manager of the Software Reuse Empirical Studies Group at the Software Productivity Consortium. Previously, he was a Distinguished Member of the Technical Staff and supervisor of the Intelligent Systems Research Group at AT&T Bell Laboratories. He has taught courses on software engineering, and on knowledge based systems at Rutgers and Columbia. He has an M.S. from the University of Illinois and an M.S. and Ph.D. from Syracuse University. He is author of Software Engineering in the Unix/C Environment (Prentice-Hall, 1991) and editor of Information Retrieval, Data Structures and Algorithms (Prentice-Hall, 1992).