Estimating the Potential for Reuse

Gregory W. Hislop

Drexel University &
Working Knowledge, Inc.
738 Elgin Rd.
Newtown Square, Pa 19073

email: hislopg@duvm.ocs.drexel.edu

Abstract

Software reuse is a promising idea, but surely offers more potential value to some organizations than others. How then do we estimate the potential for a given organization? This paper discusses a project that is exploring analysis of the organization's software portfolio as a way to help estimate reuse potential at the organization level.

Keywords: Software portfolio analysis, software repositories, domain analysis.

1 Introduction

Software reuse has substantial intuitive appeal. In addition, a number of organizations have reported success with formal programs of reuse. However, that does not mean that reuse is the right approach in every environment. On the contrary, it seems reasonable to assume that reuse would not be effective in some environments due to various technical or organizational factors.

This creates a difficult questions for people considering reuse: Is reuse the right approach for our organization?

It would be nice to answer that question by trying reuse and seeing what the results are. That approach works well for some software techniques such as prototyping. However, it appears that for reuse, there may not be significant payback until long after significant investment is made. This makes trial use difficult.

This paper discusses a research project aimed at taking another approach to assessing the potential for reuse in a particular organization. That approach centers on analysis of features in the organization's existing software portfolio.

2 Assumptions and Concepts

Reuse makes particular assumptions about the nature of software work. In addition, research results and reports of practical experience offer numerous insights into reuse. The following points are particularly relevant to this project:

1. Reuse is a promising idea, but general applicability is hard to determine.

   Researchers have reported encouraging results for formal reuse systems in a few production environments. However, some of the most encouraging results are based on judgement not measurement [Lane 84], and some reports offering measurements do not clearly define the measures [Lenz 87]. In addition, there are indications that several of the production environments studied may have unusual characteristics [Mats 90]. These factors make it difficult to evaluate the results.

2. Reuse requires substantial commitment.

   Adopting a formal reuse strategy has major financial and organizational implications. The financial issues include:

   (a) Reusable components are more expensive to produce [Lenz 87]. In part, this is due to the cost of domain analysis [Prie 90]. There is also overhead associated with the system for managing reusable components.

   (b) The size and useful life of components may limit the value of reuse. Larger components tend to be more specific and so less likely to be needed repeatedly [Bigg 87]. In addition, as technology changes and as the organization changes, the probability that a component will be useful diminishes.

   (c) Common business practice may prevent the development of component collections large enough to support a viable level of reuse [Luba 86]. Ratcliffe concluded that ".. the whole western economic system may be against reuse" [Ratc 86].

   If sharing components is not a realistic alternative, perhaps reuse is a viable strategy for only the small number of organizations that have very, very large software portfolios or very high recurrence of software needs.

   There are also organizational issues:

   (a) Software workers may resist reuse. Cavaliere and Lenz both encountered this resistence [Cava 83, Lenz 87].

   (b) Reuse affects the many aspects of software work. Meyers relates reuse and extendibility [Meye 87]. Basili relates reuse and maintenance [Basi 90]. On a different level, Tracz notes that software reuse effects elegance, quality, and discipline of software work [Trac 88]. In short, reuse affects what we do, what the artifact looks like, and how we think about the process.

Reuse requires substantial management and financial commitment [Bigg 89]. Any organization considering a formal reuse approach must weigh this commitment against potential benefits that are not clear overall and harder still to predict within the context of a single organization.

3. Reuse assumes recurring need for software artifacts.

   The potential value of reuse depends on the amount of recurrent need. If a high percent of all software work is repeated, the potential value of reuse is higher. If the percent is low, the potential is lower too.

   The potential value of reuse also depends on the exact nature of similarity and difference each time a "similar" need recurs. Differences require adjusting the existing artifact for the new need. This adjustment lowers the value of reuse.

4. Software portfolios may provide a way to explore reuse potential.

   Predicting recurrent need for software requires that we know what software will be requested in the future. While we do not know the future, we do know what needs were met in the past. We can use our knowledge of the past to suggest the nature of future needs.

   This study uses software portfolios as a representation of past software needs. Analysis of the portfolio should provide indication of both the level of recurrence and the nature of differences.

3 Related Work

This study focuses on software features that help to uniquely determine each element of a software portfolio (and to understand the similarity and difference between elements). We can divide these features broadly into form and function. Form addresses how the software looks. It includes concepts of size, complexity, control structure, data structure, and style. Function addresses what the software does. It is essentially an abstract or external view.

The importance of function is widely recognized in reuse research. So much so in fact, that similarity and functional similarity are often equated in reuse discussions [Good 83, Lane 84]. Also, prototype reuse systems, often use function as the primary means to identify candidates for reuse [Burt 87, Katz 87].

Software form is also important to reuse. It is important to note that similar function does not necessarily imply similar form [Duns 80]. However, form affects ability to understand and to modify components [Oman 88]. Both of these processes are crucial to reuse.

Since both form and function have substantial impact on potential for reuse, this study considers both elements.

1. Studies of software form.

   The study of software form has a long history, especially in the area of software metrics. Even early investigations of software form offer profiles of portfolios [Knut 71, Elsh 76].

   Some more recent studies of portfolios directly consider reuse and recurrent portfolio features. Selby applied software metrics to determine that reused modules in a portfolio had a distinct

profile [Selb 89]. Caldiera and Basili used software metrics to identify candidates to populate a reuse repository [Cald 91].

Traditional metrics are useful for identifying certain features of reused modules, they are not good discriminators of overall module similarity and difference. Similar modules may have very different metric values and different modules may have very similar metric values. Some newer metrics attempt to provide this discriminatory power [Whal 90]. These will provide a key part of the measurement of software form in this study.

2. Studies of software function.

   Other studies of software portfolios have looked for recurrent function. Goodell reports a study of this type in which the researchers attempted to develop empirically a classification scheme for business programs [Good 83]. Also, Lanergan and Grasso [Lane 84]. conducted a software portfolio study that classified a large number of programs by function.

   These reports are encouraging. They directly examine the idea of recurrent features in software portfolios, and they both find some indication of recurrent function. On the other hand, both studies were exploratory and leave many questions unanswered.

   For instance, the categorization is fairly simple. It is difficult to evaluate how relevant recurrence within broad categories is to reuse. In addition, there was no allowance for multiple functions within a program. Finally, there is no data about dimensions of program form. Are programs in the same functional category similar in form? Could they be constructed from some common component?

4 Objective and Research Questions

This study analyzes software portfolios, viewing them as a record of past software needs. The objective is to see if the level and pattern of recurrence in selected features offers insight into reuse potential.

This study analyzes features of software related to both function and form. Specific questions that the study addresses are:

1. Recurrent Function: How often do functions recur in portfolios? What functions recur often? Is the pattern of recurrence similar to that reported by previous studies?

2. Recurrent Form: How much recurrent form is there in software portfolios? What aspects of form recur often?

3. Form and Function: Is there any discernable relationship between recurrent functions and recurrent forms?

5 Status

This project is still in the preliminary stages. Current activities include selection of software portfolios, ad hoc analysis of candidate metrics for software features, and tool development.

The project is being conducted at Drexel University under the sponsorship of the Center for Multidisciplinary I/S Engineering.

References

[Basi 90] Basili, Victor R. (1990) "Viewing Maintenance as Reuse-oriented Software Development." IEEE Software. pp. 19-25. January.

[Bigg 87] Biggerstaff, Ted & Charles Richter. (1987) "Reusability Framework, Assessment, and Directions." IEEE Software. pp. 41-49. March.

[Bigg 89] Biggerstaff, Ted and Alan Perlis (eds.). (1989) Software Reusability, Vol. I. New York: ACM Press.

[Burt 87] Burton, Bruce A., et al. (1987) "The Reusable Software Library." IEEE Software. pp. 25-32. July.

[Cald 91] Caldiera, Gianluigi and Victor R. Basili. (1991) "Identifying and Qualifying Reusable Software Components". IEEE Computer. February.

[Cava 83] Cavaliere, Michael J. (1983) "Reusable Code at the Hartford Insurance Group." in Biggerstaff, Ted and Alan Perlis (eds.). (1989) Software Reusability, Vol. II. New York: ACM Press. pp. 131-141.

[Duns 80] Dunsmore, H.E., and J.D. Gannon. (1980) "Analysis of the Effect of Programming Factors on Programming Effort". Journal of Systems and Software. Vol. 1. pp. 141-153.

[Elsh 76] Elshoff, James L. (1976) "An Analysis of Some Commercial PL/1 Programs." IEEE Trans. on Software Engineering. pp. 113-120.

[Good 83] Goodell, Mike. (1983) "What Business Programs Do: Recurring Functions in a Sample of Commercial Applications". ITT: Proceedings of the Workshop on Reusability in Programming. Newport, RI.

[Katz 87] Katz, Shmuel & Charles A. Richter, & Khe-Sing The. (1987) "Paris: A System for Reusing Partially Interpreted Schemas." Also in 9th International Conf. on Software Engineering. ACM. March.

[Knut 71] Knuth, Donald E. (1971) "An Empirical Study of FORTRAN Programs." Software - Practice and Experience. v. 1. pp. 105-133.

[Lane 84] Lanergan, Robert G. and Charles A. Grasso. (1984) "Software Engineering with Reusable Designs and Code." in [Big89b] pp. 187-195. Also in IEEE Trans. on Software Eng. SE-10,5. September.

[Lenz 87] Lenz, Manfred et al. (1987) "Software Reuse Through Building Blocks." IEEE Software. pp. 34-42. July.

[Luba 86] Lubars, Michael D. (1986) "Code Reusability in the Large Versus Code Reusability in the Small." ACM Sigsoft Software Eng. Notes. 11,1 pp. 21 -27. January.

[Mats 90] Matsumura, Kazuo, et al. (1990) "Modeling of Software Reusable Component Approach and its Case Study." Proc. COMPSAC. IEEE Computer Society Press.

[Meye 87] Meyer, Bertrand (1987) "Reusability: The Case for Object-oriented Design." in [Big89b] pp. 1-33.

[Oman 88] Oman, Paul W. & Curtis R. Cook. (1988) "A Paradigm for Programming Style Research." ACM Sigplan Notices. 23,12 pp. 69-78. December.

[Prie 90] Prieto-Diaz, Ruben. (1990) "Implementing Faceted Classification for Software Reuse." Proc. 12th Int'l Conf. on Software Engineering. IEEE Computer Society Press. pp. 300-304. April.

[Ratc 86] Ratcliffe, M. (1986) "Report on a Workshop on Software Reuse held in Hereford, UK on 1,2 May 1986." SIGSOFT Software Engineering Notes. 12,1 pp. 42-47. January.

[Selb 89] Selby, Richard W. (1989) "Quantitative Studies of Software Reuse." in [Big89b] pp. 213-233.

[Trac 88] Tracz, Will. (1988) "Software Reuse Maxims" ACM Sigsoft Software Engineering Notes. 13,4 pp. 28-31. October.

[Whal 90] Whale, Geoff. (1990) "Software Metrics and Plagiarism Detection." J. Systems Software. 13 pp. 131-138.