ON THE NEED FOR TOOLS TO SUPPORT SEARCH IN SOFTWARE REUSE
A Perspective Paper Presented to Seer Technologies, Inc.

by

Tomas Isakowitz
Department of Information Systems
Stern School of Business
New York University
New York, NY 10012-1126

Robert J. Kauffman
Department of Information Systems
Stern School of Business
New York University
New York, NY 10012-1126

June 1992

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-92-41

ON THE NEED FOR TOOLS TO SUPPORT SEARCH IN SOFTWARE REUSE

A Perspective Paper Presented to Seer Technologies, Inc.

June 1992

TOMAS ISAKOWITZ

Assistant Professor of Information Systems
Stern School of Business
New York University

ROBERT J. KAUFFMAN

Associate Professor of Information Systems
Stern School of Business
New York University

ABSTRACT

Software reuse in the presence of a repository and object-based CASE tool is likely to be "biased." Prior research (Banker, Kauffman and Zweig, 1991) showed that a developer will be: most likely to reuse her own objects; somewhat less likely to reuse objects developed by her project team members; and, even less likely to reuse objects stored in the repository, but developed elsewhere in the corporation. This paper characterizes this problem in terms of three *familiarity biases:* personal bias, project bias and time bias. In the presence of these biases it is appropriate to deploy tools that support the search for software reuse, so that they may be overcome. However, the tools that are chosen or created for this purpose must adequately treat the technical and cognitive fundamentals for individual developers, and recognize the organizational and economic perspectives of a firm that wishes to maximize the business value of its software development activities.

1. INTRODUCTION

Carma McClure, a noted author and consultant in the area of software development, has recently characterized the ongoing revolution in software engineering methods in terms of three "R's", re-engineering, repository and reusability (McClure, 1992):

(1) *Re-engineering* applies capital intensive technology to support the set of activities that are involved in maintaining existing software applications. Very often re-engineering involves restructuring and optimizing existing code, reverse engineering software, and providing control as enhanced software is migrated to improve the functionality of business information systems.

(2) The *repository* is central to software automation, and provides a basis for the development tool set, life cycle integration, business function and process modeling and data architecture design, software construction and software reusability. The repository is the storehouse of a firm's software assets, which become available to other developers as they are placed in the repository, and these create value for the firm when they are combined to support revenue-generating business functions.

(3) Software *reusability* is a software development methodology that emphasizes the use of existing software assets to speed and streamline the creation of new software applications. Reuse is made possible when an automated software engineering environment supports a repository existing software assets whose functionality can be identified for reuse in new applications.

Re-engineering approaches, repository-based computer aided software engineering (CASE) tools, and software reusability represent the new frontier of software development methods aimed at achieving improved software development performance. Repository-based integrated CASE enables software reuse to extend into all the software development life cycle phases, beginning as early as functional design. Re-engineering tools, meanwhile, enable software reuse to occur in maintenance activities and in construction the first time around.

Software development methodologies that emphasize reusability are increasingly recognized in terms of the value that they deliver in helping firms to achieve higher levels of software development productivity and reduced software costs (Apte, Sankar, Thakur and Turner, 1990; Banker and Kauffman, 1991; Karimi, 1991; Kim and Stohr, 1991). Although software reuse is unlikely, by itself, to forestall the software development crisis, the attention that it has received is no doubt warranted. If firms are able to reduce the proportion of new code that must be constructed from 70-100% of the total (as in traditionally developed applications) to between just 30-40% (as has recently been observed in CASE development (Banker and Kauffman, 1991 and 1992)), the process of software development will be irrevocably altered.

1.1. Promoting and Supporting Software Reuse

In order to accomplish this, however, capital investment in tools that appropriately support and promote software reuse must occur. The research questions that we will address in this paper focus on what is meant by the words "appropriately support and promote software reuse." For a tool to offer appropriate support it must match both the *technical and cognitive perspectives of the developer*, as well as the *organizational and economic perspectives of the firm*.

Developer Perspectives. The *technical perspective* of the developer can be characterized by answering such questions as: What existing software is available for reuse, and can it be incorporated into a new application? Does the existing software match the need for specific functionality? If not, how must the existing software be modified?

The *cognitive perspective* of the developer reveals another set of concerns. These include questions such as the following: How can targets for reuse be identified? How will the set of potential targets be screened? How hard, costly or time-consuming will they be to locate? How can one tell that a software object that is targeted for reuse will really deliver the desired function? If the functionality match between what is needed and what seems to be available is not perfect, when should a developer stop searching and start building new functionality?

Firm Perspectives. The firm's perspectives differ substantially. At the level of the firm, tools that appropriately support software reuse must address basic organizational and economic concerns. From an *organizational perspective* a number of questions are raised: Can software reuse tools be deployed that will create a common environment for development to proceed? Can developers be trained to use the tools in a reasonable amount of time with predictable results? What will it take to convince developers to utilize the reuse support tools as they were intended to be used?

Finally, the *economic perspective* of the firm will require management to ask the following questions: How much will it cost to deploy reusable software support tools and how long will it take to obtain the desired results? Can the new tools be piggy-backed onto existing capabilities to minimize deployment costs? How large will the resulting impacts on development performance be and will the impacts be sustainable?

These questions set the broader context within which questions can be addressed about the appropriateness of a tool set that supports and promotes software reusability. This paper provides a basis for specifying the requirements of a software reuse support tool that can address the technical and cognitive concerns of the developer, without losing sight of the organizational and economic concerns of the firm.

2. WHY SUPPORT SEARCH FOR REUSABLE SOFTWARE?

The rationale for providing a tool to improve the effectiveness of a developer's search for reusable software follows from a consideration of several key questions:

(1) How can reuse assist in the improvement of software development productivity?

(2) What factors affect software reuse that are addressable through a reuse support tool?

(3) To what extent do search costs matter?

(4) How do familiarity biases influence a developer's search for reusable software?

2.1. How Can Reuse Lead to Improved Productivity?

In related research, we reported that reuse levels for an integrated CASE development environment deployed at the First Boston Corporation and Carter Hawley Hale Stores, Inc. contributed to higher development productivity (Banker and Kauffman, 1991 and 1992; Banker, Kauffman and Zweig, 1991). The metric that we used to gauge software reuse is called "reuse leverage" and is defined as the ratio of the number of calls made to software objects in an application and the number of new objects built specifically for the application. The metric is meant to characterize how many times an object is used on average.

Experimental development of a number of small, but realistic applications evidenced a reuse leverage ratio on the order of 3 times. In large-scale development, this level of reuse was often exceeded, rising as high as 4.11 times. A reuse leverage ratio of 4.11 times is consistent with an application that has just 24% of its functionality developed specifically for the application, while the remaining 76% is obtained through reuse.

2.2. What Factors Affect Software Reuse?

Banker, Kauffman and Zweig (1990) identified several factors that appear to have had a significant impact on software reuse. These included:

(1) the *potential to reuse* software in applications that have been designed with the intent of promoting reuse;

(2) the *search mechanism* that is employed to locate potentially reusable software objects as development proceeds; and,

(3) the *reuse implementation mechanism* that enables a developer to incorporate previously developed software objects into applications that are under construction or that are being enhanced.

Reuse is possible in all phases of the software development cycle. In fact, it is important to keep in mind that construction usually consumes 40% or less of total life cycle costs. To the extent that CASE increases the relative proportion of effort devoted to early life cycle activities such as planning, analysis and design, software reusability in the form of reusable requirements, designs and data definitions becomes increasingly important. For this reason, it is worthwhile to consider reuse potential, the search mechanism and the reuse implementation process as they apply across the life cycle phases.

Integrated CASE tools that operate in conjunction with a centralized repository of software assets have the potential to provide computerized support for reuse. For such tools to be appropriate, they will need to address each of the factors stated above: reuse potential, reuse search and reuse implementation.

2.3. Do Search Costs Matter?

When business analysts and software designers have laid out plans for software that offer the potential for reusability, the burden of reusing software will rest with developers who perform activities associated with the technical design and software construction phases of the life cycle. In the technical design phase, a developer must actually determine whether it is feasible to reuse existing software objects; in the construction phase, the existing software must be plugged into the newly constructed application.

The technical aspects of reuse will pose major concerns to developers involved in technical design. In order to reuse an object, it must be available to a developer within the repository. Firms that are actively pursuing software development in repository-based CASE normally have multiple repositories, including one for software that is under development, another for software that is being checked and tested for migration from one location to another, and a third for implemented software. Typically the development repository offers the most complete set of potentially reusable objects, but this may contain so many objects that even experienced developers will not be aware of the breadth of the functionality available for reuse.

Thus, a cost-effective search mechanism is needed to support the search for reuse. Search costs are undoubtedly a major factor influencing the observed levels of reuse leverage in a project. When search costs are unacceptably high -- for example, in the absence of a repository or well-organized code library -- it is likely that developers will search no more than the contents of their own memories. Although such search may yield considerable reuse, it is likely that a significant number of opportunities to reuse software objects will be missed.

Banker, Kauffman and Zweig (1991) reported that reuse levels at the firms whose software development operations they investigated seem to have remained constant over time, despite substantial growth in the number of repository objects and increasing programmer experience with the tool. The following facts describe why this may have been observed:

(1) 60% of software reuse involved objects written and reused by the same developer;

(2) 85-90% of software reuse involved objects that were reused by members of a project team within the same application;

(3) 5% of the developers accounted for about 20% of the software objects and over 50% of the reuse; and,

(4) the top reusers were also experience programmers, and they were able to achieve average reuse leverage levels of about 4 times, indicating that 75% of the code that they produced resulted from reuse.

Apparently a software developer is predisposed to reuse either her own software objects or those of people with whom she works closely. There is a good chance that she won't take the time to conduct a careful search of the repository to identify those objects; she will merely search her memory of for relevant software objects that were encountered in her prior development experience. This suggests that reuse is a skill that can be learned, and that top-notch developers may be most suited to creating reusable software. When there are no specific incentives to reuse software or when tools that support reuse are not available to a developer, a lower level of reuse is likely to result.

Characteristically similar results were obtained by Woodfield, Embley and Scott (1987), who examined the performance of programmers who were relatively untrained in reuse. Although they limited their examination of reuse to abstract data types stored in a software component library, the results suggested that software developers found it hard to gauge the worth of reuse, that individual biases can influence what elements are thought to be "important" in identifying targets for reuse, and that if the effort to reuse is perceived to be less than 70% of the effort to build similar functionality, then the reuse candidate was chosen. The fact that the evidence suggested that the reuse effort threshold was not at 100% (or thereabouts) of development effort is suggestive of the inefficiency of the reuse search support mechanism. (See also Fischer (1987) for additional evidence.)

2.4. How Do Familiarity Biases in the Search Mechanism Influence Reuse Search?

When a developer limits her search to just her own memory, the pool of objects that are available for reuse is constrained by three *familiarity biases*:

(1) A *personal bias* results in the developer limiting search to just his own objects.

(2) A *project bias* results in the developer limiting search to the current project's objects.

(3) A *time bias* results in the developer focusing the search on objects which have been created or reused recently, and thus are fresh in the developer's memory.

Bias in search seems to explain the observed bias in reuse. This is probably true when over half of all reuse results from a developer reusing her own objects, and when programmers who are the largest producers of software also exhibit the highest reuse levels. When 85-90% of software reuse involves objects within the same application, it is reasonable to consider the project and time biases as the factors that drive this result.

3. THE NEED FOR A TOOL TO SUPPORT REUSE SEARCH

The reuse search support tool used by the firms discussed by Banker, Kauffman and Zweig (1991) provided little more than "keyword search." Keyword search has been found to offer limited power, and be impractical in many kinds of applications (Bates, 1986; Fidel, 1985; Furnas, Landauer, Gomez and Dumais, 1987; Tarr and Borko, 1974; Zunde and Dexter, 1969). In software development, finding an object that can be reused may often require more effort than programmers are willing to expend, given the relative ease of writing the code for a single new object.

A technical tool for search should support the identification and reuse of objects beyond the boundaries of the familiarity biases which all developers are likely to exhibit. In addition to the most straightforward alternative -- full text search -- a number of alternative mechanisms to achieve this goal have been considered to date:

(1) Prieto-Diaz and Freeman (1987) proposed a object indexing scheme that

they called "facet classification", which draws on concepts in library science. Facet classification "relies not on breaking down a universe, but on building up or synthesizing" from the content of software objects (p. 8). This view matches well the perspective that is used in the object-oriented paradigm.

(2) An alternative, but related approach involves indexing software objects through "latent semantic analysis" (Deerwester, Dumais, Fumas, Landauer and Harshman, 1990). The basic idea as it relates to software reuse is that it might be possible to take advantage of the implicit or higher order structure in associating software objects with their functionality contents. "Semantic structure", according to the authors, can be exploited so that it might be possible to identify potentially reusable software objects from a cluster of characteristics, rather than single-valued descriptions.

(3) Creech, Freeze and Griss (1991) reported on an exploratory use of "hypertext search" to identify reusable elements in the context of software development at the Hewlett Packard Corporation. Similar efforts also have been under way in Europe (Vassiliou, 1990). Hypertext capabilities can be used to search a repository in ways that circumvent the constraints of less powerful search approaches, by creating links between repository objects that will help a developer to more rapidly identify the relevant objects to target for reuse.

In the latter case, even though the capabilities of the hypertext-based reuse search support tool are far more powerful than that those of keyword search, the approach has not been widely used. Hypertext tools still are in their infancy, and people who have used them in various settings report that this makes them more difficult to use than is really desirable. Faced with an opportunity to use such tools in software development, a key concern will be whether the support for reuse that is provided is cost-effective.

Clearly, the requirements for a more effective search mechanism will involve balancing the treatment of multiple aspects of the problem. A technical solution is likely to address well the technical concerns of the developer. But it may fail if it does not adequately address his cognitive concerns, and also address the organizational and economic concerns of the firm to cost effectively increase levels of reuse. Whatever mechanism is selected to support the search for reuse also must be able to span the familiarity boundary. This calls for the formulation of a mental model of the search process that can bring the familiarity boundary into sharper relief.

In future research, we intend to explore research questions that can deepen our understanding of how to support the search for reusable software. These questions include:

(1) How can we create a formal model that represents the process of search for reusable software?

(2) What is wrong with the search process used by developers who utilize currently available reuse support tools? How is its power limited?

(3) How can search for reusable software be more effectively assisted? Are there opportunities to design a technical environment that can help developers to overcome their familiarity biases in a way that is cost-effective for the firm?

REFERENCES

Apte, U., Sankar, C. S., Thakur, M, and Turner, J. Reusability Strategy for Development of Information Systems: Implementation Experience of a Bank. *MIS Quarterly*, December 1990, 421-431.

Banker, R. D, and Kauffman, R. J. "Reuse and Productivity: An Empirical Study of Integrated Computer Aided Software Engineering (ICASE) Technology at the First Boston Corporation," *MIS Quarterly*, September 1991.

Banker, R. D., and Kauffman, R. J. "Measuring the Development Performance of Integrated

Computer Aided Software Engineering (I-CASE): A Synthesis of Field Study Results from the First Boston Corporation, in *Software Engineering Economics*, T. Gulledge (ed.), Springer-Verlag Publishers, New York, NY, 1992.

Banker, R. D., Kauffman, R. J., and Zweig, D. "Factors Affecting Code Reuse," Working Paper, Center for Research on Information Systems, Stern School of Business, New York University, 1990.

Banker, R. D., Kauffman, R. J., and Zweig, D. "Monitoring the Software Asset," Working Paper, Center for Research on Information Systems, Stern School of Business, New York University, 1991.

Bates, M. J. "Subject Access in Online Catalogs: A Design Model", *Journal of the American Society of Information Science*, Vol. 37, 1986, 357-376.

Creech, M. L., Freeze, D. F., and M. L. Griss, "Using Hypertext in Selecting Reusable Software Components," *Hypertext '91 Proceedings*, ACM Press, San Antonio, Texas, December, 1991, 25-38.

Deerwester, S., Dumais, S., Furnas, G.W., Landauer, T. K., and R. Harshman . "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, Vol. 41, No. 6, September 1990, 391-407.

Fidel, R. "Individual Variability in Online Searching Behavior," *Proceedings of the American Society of Information Science 48th Annual Meeting*, Vol. 22, 1985, 69-72.

Fischer, G. "Cognitive View of Reuse Design," *IEEE Software*, July 1987, pp. 60- 72.

Furnas, G. W., Landauer, T. K, Gomez, L. M., and S. T. Dumais, "The Vocabulary Problem in Human-system Communications," *Communications of the ACM*, Vol. 30, 1987, 964-971.

Karimi, J. "An Asset-Based Systems Development Approach to Software Reusability." *MIS Quarterly*, June 1990, 179-198.

Kim, Y., and Stohr, E. "Software Reuse: Issues and Research Directions," *Proceedings of the Hawaii International Conference on System Sciences*, IEEE Computer Society Press, Vol. IV, 612-623, 1992.

McClure, C. *The Three R's of Software Automation: Re-engineering, Repository and Reusability*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1992.

Prieto-Diaz, R., and Freeman, P. "Classifying Software for Reusability," *IEEE Software*, January 1987, 6-16.

Tarr, D., and H. Borko, "Factors Influencing Inter-indexer Consistency", *Proceedings of the American Society for Information Science 37th Annual Meeting*, Vol. 11, 1974, 50-55.

Vassiliou, Y. Personal communication, November 1990.

Woodfield, S. N., Embley, D. W., and Scott, D. T. "Can Programmers Reuse Software," *IEEE Software*, January 1987; 52-59.

Zunde, P., and M. E. Dexter. "Indexing Consistency and Quality", *American Documentation*, Vol. 20, No. 3, July 1969, 259-264.