FACTORS AFFECTING CODE REUSE: IMPLICATIONS FOR A MODEL OF COMPUTER AIDED SOFTWARE ENGINEERING DEVELOPMENT PERFORMANCE

Rajiv D. Banker
Robert J. Kauffman
Dani Zweig

Department of Information, Operations, and Management Sciences
Leonard N. Stern School of Business, New York University
44 West 4th Street, New York, NY 10012

FACTORS AFFECTING CODE REUSE: IMPLICATIONS FOR A MODEL OF COMPUTER AIDED SOFTWARE ENGINEERING DEVELOPMENT PERFORMANCE

by

Rajiv D. Banker
Carlson School of Business
University of Minnesota
Minneapolis, Minnesota 55455

Robert J. Kauffman
Leonard N. Stern School of Business
New York University
New York, New York 10003

and

Dani Zweig
Department of Administrative Sciences
Naval Postgraduate School
Monterey, California 93943

December, 1990

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

Working Paper Series

STERN IS-91-1

Prepared for the Workshop on Information Systems and Economics, Copenhagen, Denmark, December 1990

Center for Digital Economy Research
Stern School of Business
Working Paper IS-91-01

ABSTRACT

An examination of code reuse at a large financial institution yields insights into the process of code reuse. The software development environment -- based on an integrated CASE system -- was designed to support code reuse, but at the end of its first two years we find that programmers are not taking full advantage of the reuse opportunities which the CASE environment provides. The organization has provided technical support for code reuse, but has not made organizational adjustments, and the technical solution alone does not suffice. We also review an existing economic model of CASE development performance that incorporates code reuse, suggesting refinements that are based upon our observations. Finally, we draw some conclusions about steps that managers can take to promote code reuse.

1. CODE REUSE IN COMPUTER AIDED SOFTWARE ENGINEERING (CASE)

One of the promises of computer aided software engineering (**CASE**) technology is that it can increase development productivity by facilitating the reuse of existing code (BOUL89, MCNU89, MOAD90, POLL90, SENT90). However, this promise has not been broadly substantiated in industry, since the technology has only recently been deployed, nor in software development performance research, which has only recently begun to examine CASE-based development platforms (KEME89, NORM89, NUNA89, SCAC87, SENN90).[1]

In recent field work on CASE technology at a large financial institution, Banker and Kauffman (BANK91) found support for the incorporation of code reuse within a model of software development performance, as a factor influencing labor productivity. However, in followup research we have obtained evidence -- both anecdotal and statistical -- that suggests that opportunities to reuse existing code may not be fully exploited, despite the real gains in productivity realized by the firm. This finding, if it can be substantiated in a broader context, points to organizational adjustments which must be made before the benefits of CASE technology with respect to code reuse can be fully realized.

We will examine a number of research questions related to code reuse that stem from these findings:

* What factors influence the level of code reuse observed in a maturing CASE development environment?
* What factors determine whether or not programmers will seek out code reuse opportunities?
* Do levels of reuse depend upon the nature of the application environment? If so, in what way?
* Beyond purely technical factors, how is code reuse affected by the managerial and organizational environment in which CASE is deployed?
* Finally, how can our answers be used to refine existing models of CASE development performance that incorporate measurements for reuse?

The remainder of the paper is organized as follows. Section 2 presents preliminary findings on code reuse within a CASE development environment. Section 3 examines organizational factors which might influence the somewhat disappointing findings reported in section 2. Section 4 concludes the paper by utilizing our results to propose an extension to a model for CASE development performance and evaluate some managerial actions that can be taken to promote code reuse.

---

[1]The reader interested in obtaining additional background on code reuse would benefit from looking at four recent papers which bring the literature up to date: Karimi (KARI90), Hall (HALL87), Seppanen (SEPP87) and Banker, Fisher, Kauffman, Wright and Zweig (BANK90). For an older (circa 1984), but still useful, examination of the state-of-the-art in software reusability, see the *Special Issue on Software Reusability* of the *IEEE Transactions on Software Engineering*, September 1984. This issue contains articles with an overview of the statistics available at that time on reuse (BIGG84, JONE84) and technical strategies to promote reuse (HORO84, KERN84, LANE84, MATS84, POLS84.)

2. THE POTENTIAL FOR CODE REUSE IN A MATURING CASE ENVIRONMENT

2.1. An Integrated CASE Environment (ICE)

The research reported here is being carried out at a large financial institution. Although the application environment is a typical one for an investment bank, the development environment is not. The software development environment, ICE -- our acronym for the integrated CASE technology deployed at the financial institution -- was deliberately designed with code reuse as an objective. The performance of this CASE environment can give us insights into the process of code reuse.

At the core of ICE is a fourth-generation language of object-oriented design. (It is not fully object-oriented, lacking inheritance in particular.) Most of the functionality of the application systems can be written in the Rule Sets of this language, and then automatically compiled for mainframes, minicomputers or workstations -- environments which previously required different languages and separate programming teams. This report will focus upon Rule Sets, but ICE application systems are built from several other object types as well, including Screen Definitions, Report Definitions, Files, Data Domains and Database Views. A Rule Set may also call an existing 3GL Module. All interactions among objects are mediated by Views. An overview of the contents of the Repository is given in Table 1 below.

INSERT TABLE 1 ABOUT HERE

All the objects of the application environment are stored in a single Repository. All the calling relationships between objects are also maintained in this Repository, as is the database. The set of relationships between objects constitutes the Repository's Metamodel. Code reuse is implemented by adding a calling relationship between a new object and one which is already in the Repository. Beyond the obvious role this capability plays in facilitating reuse, it also makes it practical to monitor reuse, without having to examine individual programs, by analyzing the relationships in the Repository (BANK90).

For the discussion which follows, we give the NUMBER_OF_UNIQUE_OBJECTS in an application system its intuitive definition. The NUMBER_OF_OBJECT_CALLS (which includes a call to the root Rule Set of the application) is the number of objects there would be, in the absence of code reuse. We define "reuse leverage" to be the average number of times each unique object is used, and compute it as (BANK90):

$$REUSE\_LEVERAGE = \frac{NUMBER\_OF\_OBJECT\_CALLS}{NUMBER\_OF\_UNIQUE\_OBJECTS}$$

For example, in Figure 1 below, there are 4 unique objects: A, B, C, and D. But, there are 5 object calls: Rule Sets B and C each call D. D would have to be replace by D1 and D2 in the absence of reuse. Reuse leverage is 5/4=1.25. (Note that without code reuse, Reuse Leverage is always 1.0, indicating that objects are used only once.)

INSERT FIGURE 1 ABOUT HERE

This measure of reuse may be adapted to traditional data processing environments.

However, in traditional environments, the data required to compute the measure may be difficult or impossible to obtain.

2.2. Object Growth in the Repository

Our expectation was that code reuse would increase over time. As time passes, the Repository grows, and so does the pool of objects which are available for reuse. In addition, not only are the opportunities for reuse growing, but so is the experience and expertise of the programmers in taking advantage of these opportunities. Presumably, this growth rate would level off with the flattening of the learning curve or with the attainment of a critical Repository size.

Figure 2 presents the growth in Rule Set[2] population and reuse during the first two years of the Repository's existence.

INSERT FIGURE 2 ABOUT HERE

It is immediately clear that the initial expectations were incorrect: The Repository grew steadily in size over this period. So did the experience of the programmers, since prior to this time ICE did not exist. Reuse Leverage, however, achieved a level of 1.4 at the beginning -- a strong showing -- but never bettered it.

2.3. Repository Reuse Demographics: Preliminary Assessment

An examination of the Repository's demographics offers us some insight into these initially surprising results. At the end of the two year start-up period, the Repository contained 8892 Rule Sets, which were called a total of 13508 times (for a Reuse Leverage of 1.5). However, although the Rule Sets were spread out over 30 major application systems (and 12 less significant systems which contained 2-19 rules each) 90% of the observed instances of code reuse involved calls between objects in the same application system. Although these Rule Sets were written by 250 different programmers, over 60% of all instances of code reuse involved calls between objects written by the same programmer.

This suggests an explanation: The opportunities for reuse may be increasing over time, but programmers are not taking advantage of them. Programmers are using objects with which they are familiar -- objects belonging to the system on which they are currently working, and especially those objects within that system which they themselves wrote. They do not appear to be searching the unfamiliar portions of the repository for reuse opportunities, but only taking advantage of those parts of which they become aware in the normal course of creating their own code.

If this is the case, we would expect to find that levels of code reuse grow with application size, since larger applications provide a larger pool of salient reuse opportunities. And, indeed, we do find corroborative evidence for this: There is a strong correlation (r=.48) between application size and code reuse.

---

[2]Rule Sets form the 'backbone' of ICE application systems. They are also the most time-consuming objects to write. (3GL modules could be more so, except that they are typically used in cases where special-purpose routines have already been written.) For these reasons, we have used Rule Sets for our initial analysis of reuse. Discussions with programmers indicate that our findings are typical of other object types.

The potential for reuse, then, is largely being ignored. Either opportunities for reuse are not being sought or, once found, they are not being used.

3. FACTORS AFFECTING CODE REUSE

We interviewed programmers to learn about the practice of code reuse within the organization. These discussions revealed some technical barriers to the realization of code reuse opportunities. Far more serious, however, are the organizational barriers and disincentives.

3.1 Search

ICE makes the invocation of a previously written object trivial. All objects reside in the same repository, and are available for reuse. The main formal mechanism for identifying such an object, however, is a keyword search mechanism, the use of which often turns out to require more effort than programmers are willing to expend. (We have found no indications that developers are not entering keywords into the index. It appears only to be the case that such keywords do not provide a sufficiently efficient search mechanism. Given the relative ease of writing any single object, programmers are often reluctant to bother with an extended search.)

3.2. Implementation

The more serious problem we identified, however, revolves around incentives. The incentive for programmers to reuse code is moderately weak. There is little managerial monitoring of reuse levels, and programmers are valued -- as is usually the case -- for their ability to meet deadlines, rather than for their ability to meet technical benchmarks. On the other hand, there are strong, informal incentives for a programmer to prevent others from reusing his or her code.

The creator of an object is its 'owner,' and every reuse of that object is a potential call upon that owner to maintain the object in case of trouble -- most likely trouble arising from its use within an environment or application for which it was not originally tuned and tested. Every reuse is also a constraint on the owner's subsequent ability to modify that object, since any modification must meet the requirements of all users of the object. (Note that incentives do not appear to be in place to motivate programmers to make their objects as general as possible in the first place.)

In practice, programmers who wish to use an object from another application are strongly encouraged (by the other programmers, not by management) to copy the object in question, to rename it, and to use it as though it were a new object. We refer to this practice as "hidden reuse," a form of reuse which is not captured by the monitoring mechanism. (It is also the dominant form of reuse in traditional applications programming environments.) It should be noted that hidden reuse achieves only some of the goals of code reuse: Coding effort and unit testing are reduced, but subsequent life cycle savings, particularly in maintenance, are not realized.

3.3. Preliminary Conclusions about Factors Affecting Code Reuse

Our initial expectations concerning code reuse rested on the assumption that the primary determinant of code reuse was reuse potential. We found, however, that for this potential to be realized, two other stages had to be passed. The potential had to be recognized, and the potential had to be used. We identified a technical barrier to the search for reuse opportunities, and a behavioral barrier to the implementation of reuse.

We note, however, that despite these barriers, the level of explicit code reuse

is much higher than has typically been seen to date in software development operations. The technical support provided by ICE is allowing at least some of the programmers to do something right. We observe further, that reuse is highly concentrated: 5% of the programmers accounted for 20% of the code and over 50% of the code reuse. Reuse leverage varies among applications from a low of 1.0 (no reuse) to a high of about 4.0. It remains to be explained what makes an application reuse-prone; what makes a programmer a re-user of code; and what practices can promote code reuse within an organization as a whole.

4. OPENING THE BLACK BOX OF CODE REUSE

Our initial expectations were based upon a previously developed model of software development productivity within a CASE environment (BANK91). In this model (shown in Figure 3) labor costs depended upon project size, programmer experience, and the level of code reuse. The level of code reuse, in turn, was taken to depend upon the opportunities offered by the particular application, and the experience level of the programmers trying to exploit these opportunities. The findings outlined in this paper, however, suggest the need for a richer model of code reuse.

INSERT FIGURE 3 ABOUT HERE

4.1. Reuse Potential, Search and Implementation

There are three points of attack for an organization wishing to promote code reuse:

* **Potential:** Systems must be designed to maximize the *opportunities* for code reuse, if they respond to encouragement to seek them out.

* **Search:** Programmers must be able to *find* these opportunities. They must also be encouraged to do so.

* **Implementation:** A reuse opportunity, once found, must be realized.

At our research site, we have observed essentially *unmanaged* reuse. The technical facilities to maximize reuse potential are present, as are some technical aids to search and implementation. However, *technical* solutions alone do not suffice. We have seen that *organizational* factors must also be considered. Finally, it has been observed that ad-hoc exploitation of reuse opportunities captures only some of the potential benefits of code reuse (KARI90). To realize the full life cycle savings potential requires an *architectural* solution as well -- corporate level systems planning for reuse.[3] This leads us to propose the revised development productivity modifiers box shown in Figure 4.

INSERT FIGURE 4 AND TABLE 2 ABOUT HERE

---

[3]Karimi (1990) uses the term 'strategic,' rather than 'architectural.'

Table 2 below organizes and reports our findings in terms of our classification of the technical, architectural and organizational approaches to promoting code reuse. Factors are classified as supported by our initial study, rejected by our initial study, or speculative (indicated by "??"). We intend to examine the impact of the speculative factors further in future research.

4.2. Conclusion

Karimi (KARI90) recently asserted that code reuse is a prerequisite for order-of-magnitude gains in the productivity of CASE development operations. In this paper, we have found evidence suggesting that to achieve high levels of code reuse, and thus, highly productive software development, management should not be satisfied with entirely technical solutions. Instead, we believe that there is only so much power a high quality CASE toolset can deliver in the absence of architectural and organizational design decisions that are conducive to the reuse of code. We hope that this research will provide management with a new way to think about the problem of code reuse.

ACKNOWLEDGMENTS

We acknowledge Mark Baric, Gene Bedell, Tom Lewis and Vivek Wadhwa for the access they provided us to data on software development projects and managers' time throughout our field study of CASE development at the First Boston Corporation and Seer Technologies. We also thank Michael Oara for his assistance with some of the data used for this paper. In addition, we benefitted from the helpful comments of the participants of the Information Systems Research Seminar held at the Naval Postgraduate School, Monterey, California (November 1990). All errors in this paper are the responsibility of the authors.

REFERENCES

BANK90 Banker, R. D., Fisher, E., Kauffman, R. J., Wright, C., and Zweig, D. Automating Software Development Performance Metrics. Working Paper, Stern School of Business, New York University, September 1990.

BANK91 Banker, R. D., and Kauffman, R. J. An Empirical Study of Computer Aided Software Engineering (CASE) Technology: Productivity, Reuse and Functionality. Forthcoming in *MIS Quarterly*, 1991.

BIGG84 Biggerstaff, T., and Perlis, A. Foreword: Special Issue on Software Reusability. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 474-476.

BOUL89 Bouldin, Barbara M. CASE: Measuring Productivity -- What Are You Measuring? Why Are You Measuring It? *Software Magazine*, 9(10), August 1989, pp. 30-39.

HALL87 Hall, P. A. V. Software Components and Reuse -- Getting More Out of Your Code. *Information and Software Technology*, 29(1), January-February 1987, pp. 38-43.

HORO84 Horowitz, E., and Munson, J. B. An Expansive View of Reusable Software. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 477-487.

JONE84 Jones, T. C. Reusability in Programming: A Survey of the State of the Art. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 484-494.

KARI90 Karimi, J. An Asset-Based Systems Development Approach to Software Reusability. *MIS Quarterly*, June 1990, pp. 179-198.

KEME89 Kemerer, C. F. An Agenda For Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts. In *Proceedings of the 22nd Hawaii International Conference on Systems Sciences*, Hawaii, January 1989, pp. 219-227.

KERN84 Kernighan, B. W. The UNIX System and Software Reusability. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 513-518.

LANE84 Lanergan, R. G., and Grasso, C. A. Software Engineering with Reusable Designs and Code. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 498-501.

MATS84 Matsumoto, Y. Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 502-512.

MCNU89 McNurlin, B. Building More Flexible Systems. *I/S Analyzer*, October 1989.

MOAD90 Moad, J. The Software Revolution. *Datamation*, February 15, 1990, pp. 22-30.

NORM89 Norman, R. J., and Nunamaker, J. F. Jr. CASE Productivity Perceptions of Software Engineering Professionals. *Communications of the ACM*, 32(9), September 1989, pp. 1102-1108.

NUNA89 Nunamaker, J. F. Jr., and Chen, M. Software Productivity: A Framework of Study and an Approach to Reusable Components. In *Proceedings of the 22nd Hawaii International Conference System Sciences*, Hawaii, January 1989, pp. 959-968.

POLL90 Pollack, A. The Move to Modular Software. *New York Times*, Monday, April 23, 1990, pp. D1-2.

POLS84 Polster, F. J. Reuse of Software Through Generation of Partial Systems. *IEEE Transactions on Software Engineering*, SE-10(5), September 1984, pp. 402-416.

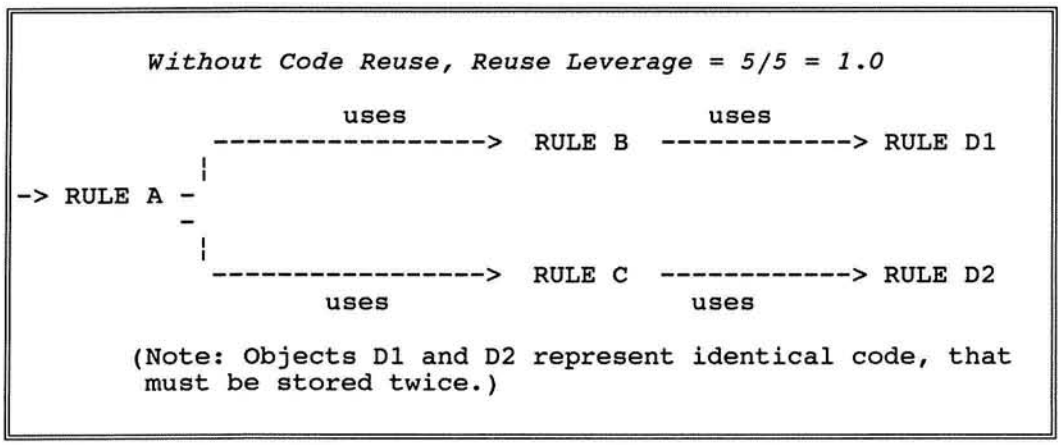
SCAC87 Scacchi, W., and Kintala, C. M. K. Understanding Software Productivity. Technical Report CRI-87-67, Computer Science Department, University of Southern California, Los Angeles, CA, 1987.

SENN90 Senn, J. A., and Wynekoop, J. L. *Computer Aided Software Engineering (CASE) in Perspective.* Working Paper, Information Technology Management Center, College of Business Administration, Georgia State University, 1990.

SENT90 Sentry Market Research. *CASE Research Report*, Westborough, MA, 1990.

SEPP87 Seppanen, V. Reusability in Software Engineering. In P. Freeman (ed.), *Tutorial: Software Reusability*. Computer Society of the IEEE, 1987, pp. 286-297.

Figure 1. An Illustration of Reuse Leverage



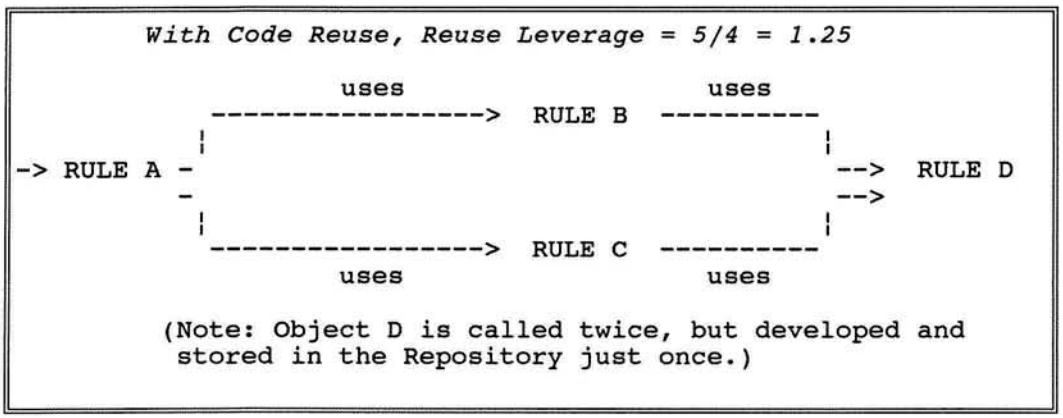
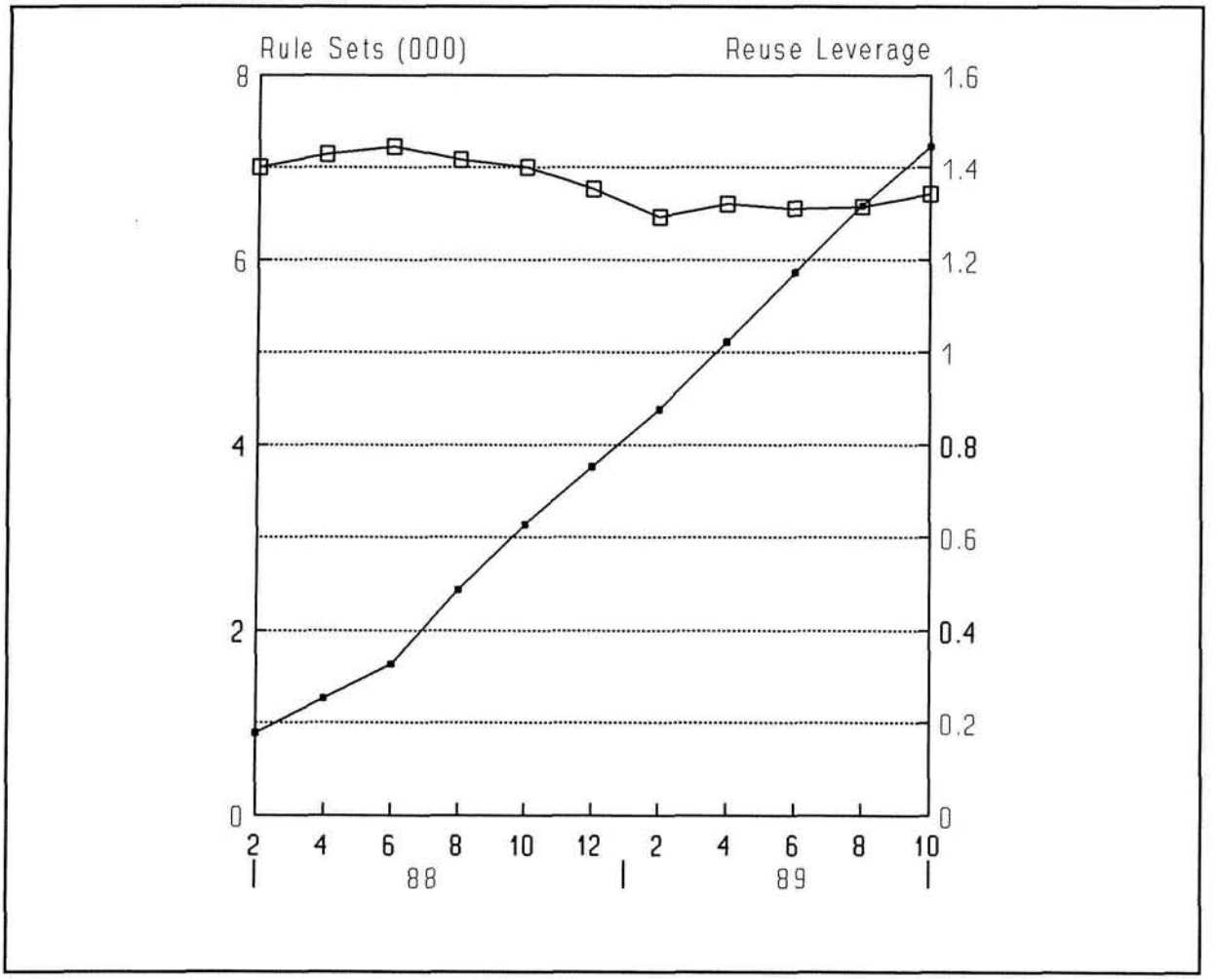


Figure 2. Code Reuse in a Maturing Repository



* Cumulative Objects
■ Reuse Leverage

Figure 3. An Economic Model of CASE Development Productivity

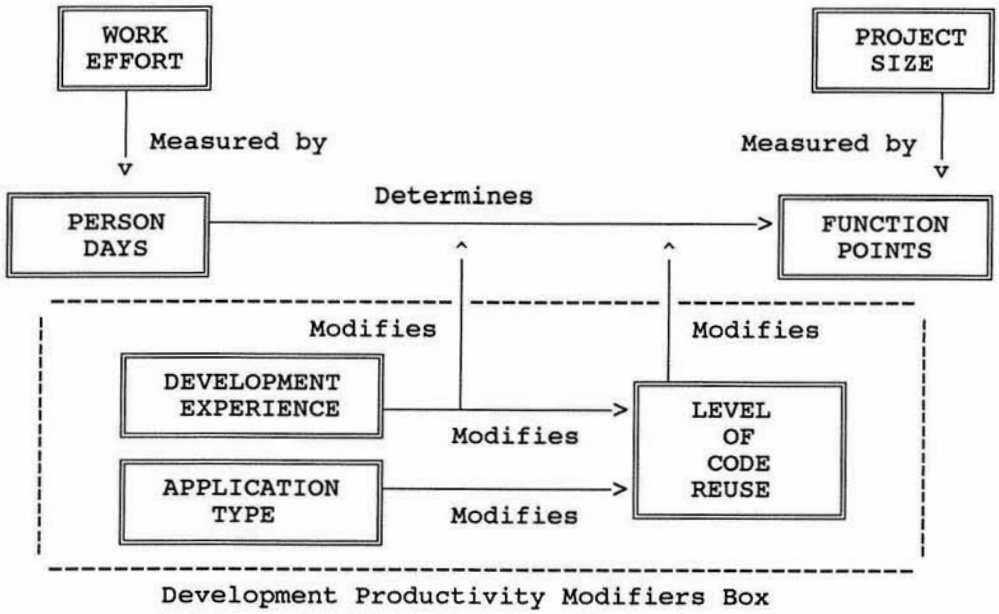


Figure 4. Revised CASE Development Productivity Modifiers

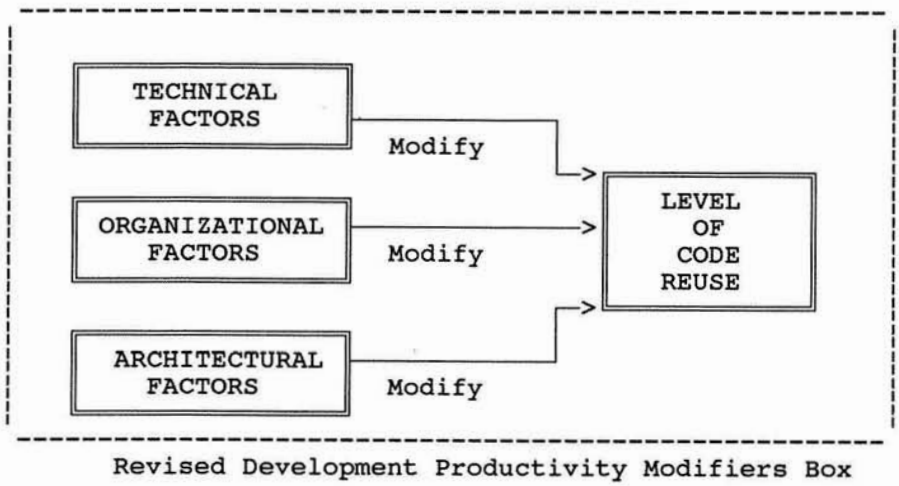


Table 1. An Overview of the ICE Repository

OBJECT TYPE	OCCURRENCES
Rule Sets	8892
Screen Definitions	7230
Domain Definitions	4200
File Definitions	4236
3GL Modules	6062
Data Fields	6266
Database Views	6755

Table 2. Factors Affecting Code Reuse: Evidence and Future Research

REUSE FACTOR CATEGORY	EXAMPLES OF SPECIFIC REUSE DRIVERS	SUPPORTED BY OUR RESEARCH?
Technical	* Object-oriented CASE tool	Yes
	* A more mature Repository	No
	* Larger applications promote reuse	Yes
	* Application type matters	Yes
	* Keyword search facilities	No
Organizational	* Incentives to reuse code	??
	* Training in code reuse	??
	* Star developers may also be experts at reuse	Yes
	* Team size managed to promote reuse	??
	* "Ownership" of Repository objects	Yes
	* Object librarian or administrator	??
	* Monitoring of reuse levels	??
	* Maintenance responsibilities	Yes
Architectural	* Application size	Yes
	* Programming guidelines to promote reuse	??
	* CASE development process designed to Promote code reuse in the analysis and design phases of the life cycle	??