# SUPPORTING SEARCH FOR REUSABLE SOFTWARE OBJECTS

**Tomas Isakowitz**

**Robert J. Kauffman**

**Information Systems Department**
**Stern School of Business**
**New York University**

Replaces IS-92-41

**Working Paper Series**
STERN IS-93-47

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

# SUPPORTING SEARCH FOR REUSABLE SOFTWARE OBJECTS

**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

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

# SUPPORTING SEARCH FOR REUSABLE SOFTWARE OBJECTS

*TOMAS ISAKOWITZ*

*ROBERT J. KAUFFMAN*

## ABSTRACT

Software reuse in the presence of a repository and object-based CASE tool is likely to be "biased." Prior research has shown that a developer will be: (1) most likely to reuse her own objects; (2) somewhat less likely to reuse objects developed by her project team members; and, (3) even less likely to reuse objects stored in the repository, but developed elsewhere in the corporation. These biases can result in sub-optimal levels of software reuse. In the presence of such biases it is appropriate to deploy tools that support the search for software reuse, so that developers find it easier to reuse software objects authored by developers other than themselves or project team members. However, the tools that are chosen or created for this purpose must adequately treat the technical and cognitive fundamentals of the problem 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. In this paper we present a two-stage descriptive model that represents the search process for reusable software objects. We evaluate appropriate technologies, propose a technical solution to the problem of searching for reusable objects, and demonstrate its feasibility via a prototype implementation. The technical tool combines an automated classifier and a hypertext system. We describe an architecture to automatically create hypertext networks based on the classification schema. We illustrate our architecture using a classification of software objects obtained through structured interviews with software developers.

## ACKNOWLEDGMENTS

The authors wish to acknowledge Marc Baric, Gene Bedell, Tom Eichner, Gig Graham, Richard Mosebach, Michael Oara, Tom Robben, Cecelia Poppleton, Vivek Wadwha and Charlie Wright for the access they provided us to data on software development projects and manager's time throughout our field study of software reuse in CASE development at Kidder Peabody and Seer Technologies. We also thank the software developers and managers for participating in interviews that enabled the research results on which we report here. We also thank Tony Phillips, Allan Rosenstein, Howard Sloan for developing the prototype system. Thanks to Michael Bieber and Hank Lucas whose comments influenced this paper. All errors are the responsibility of the authors.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

## 1. INTRODUCTION

Software development methodologies that emphasize reuse are increasingly recognized by senior management in terms of the value they deliver in helping their firms to achieve higher levels of software development productivity and reduced software costs [1], [3], [23], [25]. Although software reuse is unlikely, by itself, to forestall the software development crisis, the attention that it has received is 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 we have recently observed with CASE [3], [4] -- the process of software development will be altered for the better. In order to accomplish this, however, capital investment in tools that appropriately support and promote software reuse must occur. 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. A key ingredient to promoting software reuse is *reusable object search*, i.e., the problem of locating suitable software objects [1] to be reused. We propose a technical solution to this problem founded on a hypertext architecture based on a classification of software objects .

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 questions such as: What existing software is available for reuse, and how can it be incorporated into a new application? Does the existing software match the need for specific functionality? If not, to what extent must the existing software be modified? The *cognitive perspective* of the developer reveals another set of concerns. These include questions such as: How can objects with potential for reuse be identified? How will this set of targets be screened? How hard, costly or time-consuming will they be to locate? Will the object selected really deliver the desired function? If the functionality match is not perfect, when should a developer stop searching and start building a new object? For software reuse to be successful, developers should perceive search to be less expensive than construction, i.e., the perceived cost of discovering an existing object to fulfill a given need should be less than the perceived cost of developing it anew.

[1]The meaning of the word *object* in repository-based CASE environments differs from the meaning it takes in Object-Oriented environments. In a repository-based CASE environment, the term *software object* serves to denote repository components. In an Object-Oriented environment however, an *object* represents a domain entity by encapsulating data and functionality.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

**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? Will they deliver the kinds of reuse that are most beneficial to the firm? The *economic perspective* of the firm will require management to ask another set of questions: How much will it cost to deploy reusable software support tools? How long will it take to obtain the desired results? Can the new tools be merged onto existing capabilities to minimize deployment costs? How large will the resulting impacts on development performance be? Will the impacts be sustainable?

These questions set the broader context within which to address the appropriateness of a tool set that supports and promotes software reuse. In this paper we present a technical solution that addresses the cognitive and technical aspects of developers while addressing the firm perspectives. Our solution promotes the development of a holistic development environment that fosters reuse and fits in well within the current CASE environments.

The tool we propose is based on a combination of an automated classifier for reusable objects and a hypertext system to support the identification of objects suitable for reuse. Classification approaches to reuse have been proposed in earlier work (see for example [29]). Hypertext technology applies well to domains where information is semi-structured and relationships among domain elements are important [22]. Such is the case with software engineering, especially in CASE environments, where software objects such as code, documentation and designs, are semi-structured -- because they conform to the formal guidelines of programming language syntax and documentation style-- and are tightly interconnected (for example, programs call one another, programs use files, files and programs have documentation, etc.). Reusable object search in a CASE environment is therefore performed over a domain of semi-structured objects that are tightly interrelated according to formal guidelines, and this makes a hypertext-based solution suitable.

Section 2 examines results in prior research involving organizations that have employed reusable software development strategies and highlights that reuse fails to pay off as senior management hoped. Section 3 offers a partial answer to this problem, by reviewing and evaluating alternate reusable object search support mechanisms that will enable developers to expand their awareness of the contents of a large repository of software objects. Section 4 develops a two-stage descriptive model of the search process for reusable objects. The

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

primary argument is that search involves separate activities: identification of potential targets and functionality screening. A related argument is that each needs to be supported in a different manner to maximize effectiveness. We show how a combination of search mechanisms, including faceted classification and hypertext, can provide the necessary support for reusable object search. We illustrate these concepts by drawing on an example that was developed from structured interviews with software developers working in ICE, an Integrated Case Environment deployed at a major investment bank in New York City, that supports software reuse. In section 5 we provide a proof of concept of our approach via a prototype implementation of the reuse search facility. Section 6 concludes the paper by reviewing the main conceptual contributions of this research, presenting other aspects of related research in progress and some caveats and implementation considerations.

## 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, and what factors affect software reuse?

(2) To what extent do search costs matter?

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

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

### 2.1. Development Productivity and the Factors Affecting Software Reuse

Although reuse is possible in all phases of the software development cycle, construction usually consumes 40% or more of total life cycle costs, so it is natural that many efforts to improve software development performance through automation have focused there. However, CASE increases the relative proportion of effort devoted to early life cycle activities, such as planning, analysis and design, so software reusability in the form of reusable requirements, designs and data definitions has become increasingly important.

Banker and Kauffman [3], [4] recently reported on reuse levels for an integrated CASE development environment deployed at The First Boston Corporation and Carter

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

Hawley Hale Information Services, that contributed to higher development productivity, especially in construction. The authors measured the extent of software reuse along the lines of the emerging standards that are espoused by the *Technical Committee on Software Engineering of the IEEE Computer Society* [32]:

$$\frac{REUSED\_SOURCE\_STATEMENTS}{NEW\_SOURCE\_STATEMENTS + REUSED\_SOURCE\_STATEMENTS} * 100\%$$

The software reuse ratio is defined as (3)/ [(1)+(3)] * 100% in terms of the elements shown in Table 1.

| | **USAGE** | |
|---|---|---|
| **ORIGIN** | ***Delivered*** | ***Non-delivered*** |
| ***Developed*** | **New Source Statements (1)** | **New Source Statements (2)** |
| ***Non-Developed*** | **Reused Source Statements (3)** | **Reused Source Statements (4)** |
| | **Deleted Source Statements (5)** | **Deleted Source Statements (6)** |

**TABLE 1: IEEE STANDARD SOURCE STATEMENT ORIGIN AND USAGE MATRIX**

The IEEE standards, which deals with program code, do not apply well to CASE environments because the latter contain software objects other than program code. However, the standards can be specialized to deal with general software objects resulting in adequate standards for CASE environments. Experimental development of a number of small, but realistic repository object-based applications evidenced *object reuse percent* on the order of 67% [3], [4]. In large-scale development, this level of object reuse was often exceeded, rising as high as 76%. An object reuse percent of 76% is consistent with an application where an object is reused an average of 4.11 times. It is interesting to note that the firms at which these observations were made did not have explicit incentives in place to promote reuse, other than the motivation that a developer would have to improve her own performance. Nor were there

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

especially powerful tools in place to encourage reuse. As a result, Banker, Kauffman and Zweig [5] characterized the observed reuse levels as being a conservative estimate of what could actually be achieved if additional technical support features and new developer incentives were implemented. However, reuse levels did not exhibit significant increases as new applications were developed and developers became better at using the CASE tool. In part this can be adjudicated to difficulties in locating objects for reuse [5].

When business analysts and software designers have laid out plans for software that offers the potential for reusability, the burden of reusing existing software objects 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 reuse is feasible; 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 a software object, for example, it must be available within .he repository. Firms that are actively pursuing software development in repository-based CASE normally have multiple repositories, including the *development repository* for software that is under development, the *testing/migration repository* for software that is being checked and tested for migration from one location to another, and the *production repository* for implemented software. The development repository typically 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.

## 2.2. Search Costs and Familiarity Biases

Conventional search tools are likely to be too costly, in terms of the time it takes locate an object for reuse, to be effective in repository-based CASE environments. Search costs are undoubtedly a major factor influencing the observed reuse percent in a project. When search costs are unacceptably high -- for example, in the absence of a repository or a well-organized code library -- it is likely that developers will search no more than the contents of their own memories. Although such search still may yield considerable reuse, it is likely that a significant number of opportunities to reuse software objects will be missed.

Banker, Kauffman and Zweig [5] reported empirical results pertaining to software reuse that bear out this observation. They showed that reuse levels at two firms whose software development operations they investigated seem to have remained constant over time,

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

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 constructed 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 experienced programmers, and they were able to achieve average reuse leverage levels of about 4 times, indicating that 75% of the objects that they produced resulted from reuse.

These observations suggest that current reuse practices are affected by three kinds of *familiarity biases*:

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

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

(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.

The existence of personal biases in search is supported by two observations. First, over half of all reuse results from a developer reusing her own objects; and second, 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.

Similar results were obtained by Woodfield, Embley and Scott [35], who examined the performance of programmers 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:

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

(1) individual biases can influence what elements are thought to be important in identifying targets for reuse; and,

(2) 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.

(3) otherwise, software developers found it hard to gauge the worth of reuse, and thus the worth of their efforts to locate appropriately reusable code;

The reuse effort threshold of less than 100% (or thereabouts) of development effort suggests that the available mechanism to support the search for reusable software was not efficient. (Fischer [16] provides additional evidence for this assertion.)

When 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 will not take the time to conduct a careful search of the repository to identify for objects offering the best functionality match. When those objects exist in the repository, and the developer builds them from scratch, the reuse power of the CASE tool and the supporting repository are lost. Reuse is a skill that can be learned, but when there are no specific incentives to reuse software or when tools providing appropriate support for reusable object search are not available to a developer, a lower level of reuse is likely to result. Thus, a cost-effective search mechanism is needed to support the search for reuse.

## 3. TOOLS TO SUPPORT SEARCH FOR REUSABLE OBJECTS

There are managerial and technical approaches to support developers' search for reusable software objects. A *managerial support approach* can take the form of a group of reusability experts who advise developers on the contents of the repository so that they can find and retrieve objects of interest. Alternatively, a reuse committee can be put in charge of managing the repository. This entails the screening of objects to be included into the repository to enforce quality and boost reuse, for example, by requiring that objects be specially crafted to be reused. They can also act to reduce or eliminate redundancy by impeding the addition of objects with overlapping functionality. These managerial approaches to search have been adopted by various firms with varying degrees of success. However, in the absence of a powerful technical tool to support reuse search, managers are unable to set realistically attainable reuse level goals. A *technical support approach,* consists of computerized tools that can assist developers in identifying and retrieving objects for reuse. In

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

this section we describe four techniques: keyword search, full text retrieval, structured classification schemes and hypertext.

### 3.1. Keyword Search

*Keyword search* requires assigning to each object a number of relevant keywords or indices. As an example, consider an investment bank that has developed a number of in-house applications using a centralized repository. Within the general ledger application there is module entitled ***EDIT-ENTRY*** that enables users to edit entries stored in a file. The ***EDIT-ENTRY*** software object uses a buffer implemented as a string of characters; and it accesses a file. The following keywords can be associated with this object: *EDITING, BUFFER, STRING INSERTION, STRING DELETION, STRING CHANGE, GENERAL LEDGER DIARY, ENTRY, ACCOUNTING, FILE I/O.* Search for this object within the object repository involves the specification of a number of keywords, and the subsequent retrieval of matching objects. For example, a developer looking to implement a module to edit entries in an account receivables record could issue a search on the keywords *EDITING* and *ACCOUNTING.* The ***EDIT-ENTRY*** object would be retrieved because it has been indexed with those keywords.

A common objection to the keyword method is the high cost associated with manual indexing, which requires skilled personnel. Another objection is related to the ambiguous nature of keywords that can lead to substantial disagreement over the choice of keywords [6], [15], [17], [31], [37]. Therefore, keyword search has been found to offer limited power, and to be impractical in many kinds of applications. In our setting, keyword-based object search would require developers to provide appropriate keywords for *every* object in the repository. Interviews we held with developers in various firms showed that developers did not willingly assign keywords to the software objects they create -- there is no perceived direct benefit for the extra level of effort involved.

### 3.2. Full-text Retrieval

The high cost of manual indexing has made it attractive to automate the indexing process. The simplest kind of automatic indexing is illustrated by *full-text retrieval* systems. Such systems work on the basis of a simple mechanism:

*"Store the full text of all documents in the collection in a computer so that every character of every word in every sentence of every object can be located by the machine. Then, when a person wants information from that stored*

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

*collection, the computer is instructed to search for all documents containing certain specified words and word combinations, which the user has specified."* ([10], 289)

Full text search works best for software objects that have embedded or attached comments. Full-text retrieval systems can be made quite efficient by pre-processing the repository and constructing index tables ahead of time. Search in this case involves a table lookup, which can be done very efficiently. Speed, however, is not the only relevant criterion. Blair and Maron [10] showed that for large textual bases, full-text retrieval misses many objects -- as many as 80% of them -- relevant to the search. Therefore, this search method cannot ensure better performance than the current practice, even in the presence of familiarity biases. Hence, full-text retrieval is inadequate to support search for reusable objects.

### 3.3. Structured Classification Schemata

*Structured classification schemata* use a fixed number of predetermined perspectives, or facets, for classification. Table 2 contains sample entries from a library of software components using a six facet classification scheme due to Prieto-Diaz [29]. To search for a software component, a developer issues a query consisting of a sextuple of values that is compared to the components in the library.

A common problem with this approach lies in the handling of synonyms and of ambiguous words. An inadequate treatment of synonyms can result in the retrieval of objects irrelevant to the search. Word ambiguity can cause low retrieval rates when only few of all possible meanings of a word are considered; it can also lead to the retrieval of irrelevant objects when unintended word meanings are considered. One way of addressing these issues is to limit the vocabulary for classifying software components, and to only allow queries drawn from this controlled vocabulary. Another problem is the implementation of *near matches*, i.e., retrieving components that do not exactly match the query, but closely resemble it. To solve this problem, Prieto-Diaz proposes the use of a *conceptual graph* that determines a numeric distance between words; this system produces a metric to rank the relevance of objects to a particular query.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

| **Function** | **Objects** | **Medium** | **System Type** | **Functional Area** | **Setting** |
|---|---|---|---|---|---|
| add | array | disk | compiler | accounts payable | advertising |
| edit-entry | buffer | keyboard | editor | accounting | banking |
| measure | buffer | keyboard | code optimizer | budgeting | car dealer |

**TABLE 2: FACETS IN THE CLASSIFICATION SCHEMA OF PRIETO-DIAZ** [29]

The faceted classification of Prieto Diaz does exploit the characteristics of CASE environments, where the repository contains a wider variety of software objects as well as more detailed information about relationships among these objects, than those considered in non-CASE software libraries. For example, Prieto-Diaz' classification does not contemplate repository information: some objects are *drivers* -- they call other objects -- and some are *leaves*. The repository records whether an object uses, or is used by, other objects; whether an object uses *files* or whether it calls a *window* to converse with the user. These distinctions are finer that those contemplated in the classification schema shown below. Moreover, some of the distinctions shown in Table 2, e.g., **objects** and **medium** do not apply in the CASE domain because in CASE environments objects are represented at a level that is implementation-independent. Moreover, some of the distinctions made in Prieto Diaz' schema do not apply to CASE environments because CASE objects exhibit higher levels of abstraction than those present in the source code of third generation language libraries. For example, the facets **objects, medium** and **system type** are unlikely to be relevant in CASE environments. As we show in section 4, it is possible to adapt faceted classification to integrated CASE environments.

Faceted classification is somewhat inflexible in that it requires all objects to be classified in terms of the same facets. Synder [34] proposes a classification mechanism that allows for software objects to be classified along specialized facets without imposing these *special* facets on all software objects. This approach, based on semantic networks [36] promotes flexibility, can also be adapted to CASE environments.

A disadvantage of these classification approaches vis-a-vis full-text retrieval is that they require manual classification, as automatically deducing software functionality is quite complex. In integrated CASE environments however, the information available in the repository can be used to automatically classify software objects. Furthermore, the existence of a centralized software repository in integrated CASE environments supports access to software objects for inspection purposes. Hence these environments provide the opportunity

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

to automatically classify software objects and to support exploratory activities for software reuse. Since the classification approaches discussed here fail to exploit these potentials, an alternative -- or complementary -- approach is called for.

## 3.4. Hypertext Search

Hypertext represents one of the newest forms of computer-based support for reading documents. Rather than being constrained to the linear order of conventional documents, users are able to move through a hypertext document by following links represented on the screen by buttons. (We refer the reader to Conklin [12] and Nielsen [27] for introductions to hypertext.) The basic building blocks in hypertext are nodes and links [21]. Each *node* is associated with a unit of information, and nodes can be of different types. The node type depends on various criteria, for example, the class of data stored (plain text, graphics, audio or an executable program), or the domain object it represents (diary entry, account, financial statement). *Links* define relationships between source and destination nodes, for example, a link can connect the name of a customer in an invoice to a detailed customer profile screen. Links are accessed from the source node and can be traversed to access the destination node.

Current hypertext systems provide users with sophisticated user interface tools that enable them to inspect node contents, and to navigate through the network by selecting a path to follow. For example, clicking on the name of a customer will result in a display of his detailed profile. Besides allowing users to traverse links at their own discretion, hypertext systems provide users with pre-defined paths through the network, and with the ability to specify search conditions for the selection of nodes. Their queries may be *content-based* (searching the content of nodes, e.g., "all occurrences of the word **print**") or *structural* (depending on the topography of the hypertext network, e.g., "all software objects that have a link labeled **uses** to the **main program**"). Because a major problem with hypertext is the potential for users to get lost in the details of the information that can be accessed [28], hypertext systems usually provide backtracking and other navigation aids such as maps, to help orient the user.

Hypertext has been used previously to organize software repositories. DIF, Document Integration Facility [19], is a hypertext system to support the development, use and maintenance of large-scale systems and their life-cycle documents. Dynamic Design [8],[9] is a hypertext-based repository that organizes relationships between various software components such as specifications, design documentation, program documentation, user documentation, source code, object code and symbol tables. It enables easy access to components that are related to each other and it helps during validation by linking source code to requirements.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

Robson [30] presents DYHARD, a language that specifies how hypertext links are to be created. DYHARD can be used to develop a software tools that support hypertext-based repositories. Creech, Freeze and Griss describe *KIOSK* [11], a hypertext system to access a structured library of software components. Repository objects are represented as nodes, and relationships among the objects as links. The links record code dependencies between modules, inheritance among classes and between software objects and their documentation. In addition, KIOSK supports multiple views of the library that correspond to the various roles of those involved in software development (e.g., developers, designers, users). Each such view has its own set of links and nodes. A software developer can specify her focus with filters that restrict access to links. The ESC project [7] is an attempt at organizing software sources and documentation as a hypermedia encyclopedia to foster reuse. ESC can integrate disperse repositories by communicating with servers across a telecommunication's network.

The capability of hypertext to represent semi-structured information has been exploited to support specification and design activities; and more generally, cooperative CASE environments. For example, HyperCASE [14] and Dynamic Design [9],[8] are hypertext-based CASE environments. Conklin and Begeman's gIBIS [13] enables systems designers to collaborate by organizing discussions on system design and by capturing reasoning behind design decisions. Another example is Garg and Scacchi's ISHYS (Intelligent Software Hypertext SYstem) [18], that supports the specification phase of software development and actively checks for consistency and completeness of software specifications. Intermediary agents are proposed by Kerola and Oinas-Kukkonen [24] to facilitate interaction within a CASE environment through hypertext.

Except for ESC [7] and KIOSK [11] little research has been performed on the potential of hypertext to directly support search for reusable software objects. As Creech, Freeze and Griss report [11], KIOSK's success was limited because developers were not willing to spend the extra time required to learn to use the new facility. Similar difficulties arose within the scope of the ESC project. Our approach differs from KIOSK in two significant ways. First, the hypertext system is structured using a classification that matches the software developer's mental model of ICE. Second, the tool is to be seamlessly integrated within the ICE environment to minimize leaning costs by developers. Hence the system is likely to be natural and easy to use for ICE developers.

A helpful hypertext concept is that of a guided tour [33]. In a guided tour, navigation is usually constrained to a few choices. Although they seem to limit the power of hypertext, guided tours help users focus on a specific path and reduce the disorientation than can occur because of a large number of navigational possibilities. For example, the collection of all

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

software objects in a repository that implement a "customer SQL-update" can be organized into a guided tour. System developers then navigate among the various elements of the guided tour to locate the ones that more closely match their needs. Garzotto, Paolini and Mainetti [20] implement guided tours as *linear* paths that enable users to navigate only backwards and forwards through an ordered list of nodes. All other links are disabled when traversing a guided tour. As soon as the user exits a guided tour, the other links are re-enabled to allow for free exploration.

Hypertext technology applies well to CASE environments because the information units (e.g., software components, documentation) are quite structured and the relationships among domain elements are clearly defined, and hence lend themselves to representation as links. can be represented in a clear way. As we show in Section 4, there is potential for wider use of this technology to support search for reusable software objects.

We have reviewed four methods used to support search reuse. The first three, keyword search, full-text retrieval and structured classification schemata, represent approaches that are based on a classification of repository objects. In these approaches, search for reusable objects is implemented as a query process that employs the relevant classification. Hypertext represents a fourth approach based on a navigational metaphor. It gives users the ability to navigate the space of repository software objects at their own will. We propose combining automated classification and hypertext in a tool that provides navigational capabilities based on a classification of repository objects.

## 4. AN ARCHITECTURE TO SUPPORT REUSABLE OBJECT SEARCH

A key aspect of our design is the use of a robust classification schema. We developed such a classification for ICE based on field interviews with software developers. The classification is used to structure the repository, to assist developers in formulating queries for objects to be reused, and to process these queries. The hypertext system facilitates the detailed inspection of the set of objects that the query processor retrieved from the repository. As our prototype will demonstrate, such classification can be automated.

In order to make the search process more effective, we use a two-stage approach as shown in Figure 1. Stage 1 is *screening*; it involves the purposeful evaluation of a large set of object reuse candidates from the entire repository of software objects to determine a subset of near matches for further investigation. During Stage 2, *identification*, developers closely examine this subset to determine if any of its objects provides the desired functionality.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

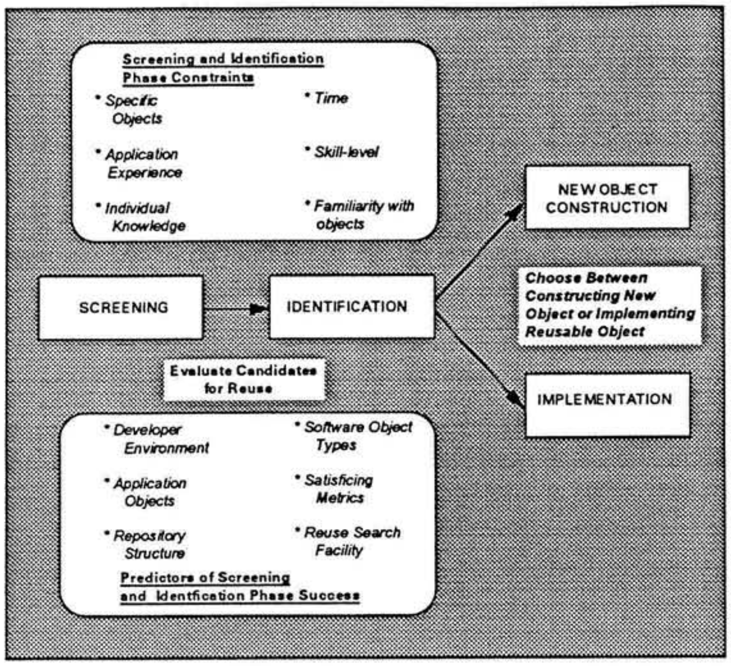


**FIGURE 1. A TWO-STAGE DESCRIPTIVE MODEL OF SEARCH FOR REUSABLE OBJECTS**

The underlying idea is that a developer's involvement in the screening process should purposely be kept to a minimum. With a large number of objects to screen -- almost like a needle in a haystack -- it is unlikely that the requisite functionality could be identified in Stage 1. On the other hand, we would expect the developer to be more proactive in Stage 2, where identification would occur from among a smaller number of relevant objects.

## 4.1. Stage 1 -- The Screening Process

Using the classification schema, the user specifies the requisite functionality that needs to be implemented. We use an interactive screen in which descriptors, belonging to various facets, are elicited from the developer. Screening consists of retrieving from the repository a set of objects that belong to the class specified by the developer. The objective during this stage is the retrieval of a sizable yet manageable set of candidates for reuse. There are two steps in this stage:

- **Step 1:** The developer enters a description of the requisite functionality by providing values for the classification facets.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

- **Step 2:** The repository is scanned using the classification schema producing a set of objects, which provide a reasonable or approximate "first-pass" match for the desired functionality.

By the end of screening, a sizable set of relevant objects -- the set of candidate objects for reuse -- has been extracted from the repository. Some of the retrieved objects may not be completely relevant to the task to be implemented. However, most of the relevant ones will be included. However, it would be too labor-intensive to examine each of them at this point: there will still be too many. Instead, the number of candidates needs to be organized to make it feasible for a developer to inspect them individually during the identification stage.

## 4.2. Stage 2 -- The Identification Process

To facilitate inspection for reuse, the set of candidate objects obtained from the screening phase is structured as a network of hypertext guided tours according to the classification schema. This will aid developers to inspect objects that exhibit similar functionalities. in Figure 2 illustrates the use of guided tours to interweave related software objects. The objects shown in the figure all agree on the value for facet *f*. Construction of the guided tours is based upon the following mechanism:

*If $a_1$, $a_2$, ..., $a_n$ are all the objects obtained from screening phase with a value of f for facet **F**, they are collected into a guided tour GT= { $a_1$, $a_2$, ..., $a_n$}. First the objects are ordered*[2]*, then links labeled f are created to connect a special start node labeled **Guided Tour f** ($GT_f$) to node $a_1$, node $a_1$ to node $a_2$, node $a_2$ to node $a_3$, and so on, closing the list by linking node $a_n$ back to the start node $GT_f$.*

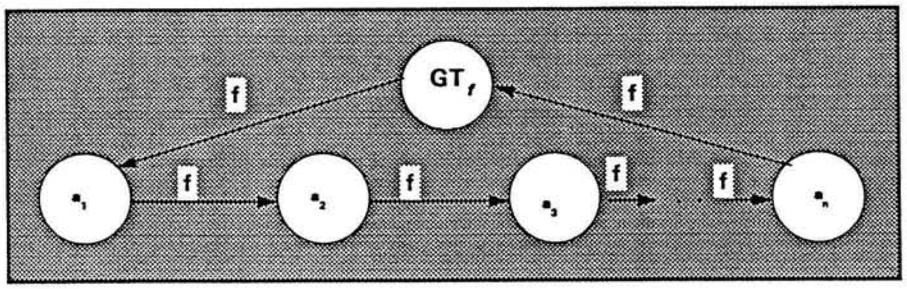


**FIGURE 2. A GUIDED TOUR *(GT)* CONNECTING THE OBJECTS WITH THE SAME VALUE (*VAL)* FOR A FACET *(F)***

[2]The ordering criteria can be arbitrary, e.g., lexicographic by rule name, or it can reflect a more developed metric if available.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

Software objects may well be shared by various guided tours, resulting in a network of interconnected links. Figure 3 shows a portion of two intersecting guided tours, one linking objects classified under value **INTERACTION** (rule 7, rule 8, rule 10, rule 11 and rule 12), the other linking objects classified as **SQL** (rule 7, rule 12 and rule 15).

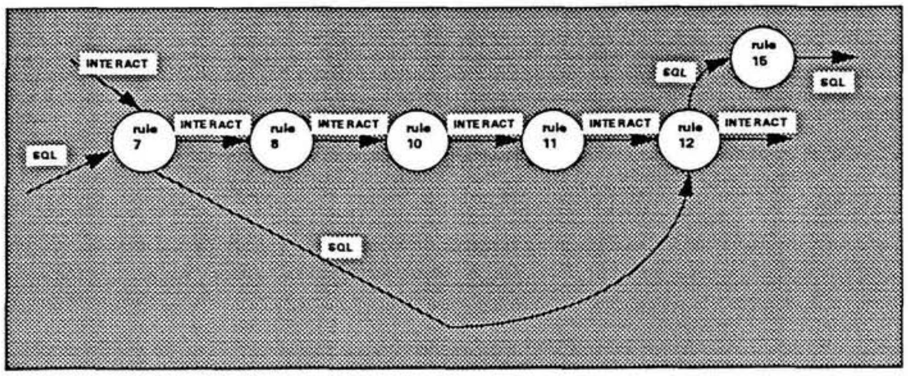


**FIGURE 3: INTERSECTING GUIDED TOURS**

Once the network has been created in this manner, the developer can proceed to explore the set of candidates in a structured manner using a hypertext-based tool that enables her to inspect various objects by navigating from object node to object node, within the set of candidates for reuse. The navigational capabilities of hypertext facilitate a rapid traversal of the network to locate target objects for reuse. The navigation is helpful in zeroing in on the requisite functionality because the links are set up according to a classification schema which reflects the mental model that software developers have of the repository.

User-driven navigation achieves multiple purposes. It enables the exploration of a relatively large set of objects, and hence, the initial screening does not need to be very precise. It also allows developers to make their own decisions. The benefits of the navigational aids provided by this hypertext-based tool are likely to have a favorable impact, reducing search time and, thus, overall search costs.

To summarize, identification consists of the following two steps:

- **Step 1:** The objects retrieved are automatically structured into guided tours, in accordance with the classification schema.
- **Step 2:** The developer proceeds to explore the candidates for reuse using the hypertext tool to inspect individual objects and to navigate among them in search of relevant objects to be reused.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

When identification concludes, the developer will have located and retrieved a small set of applicable objects that can be reused, or she will be almost certain that no applicable objects are readily available in the repository.

The ambiguity problems we referred to in section 3.3 are addressed as follows. Lexical ambiguity is reduced by using a controlled vocabulary. When developers select search criteria they do so by picking from a set of given classifiers -- a pull-down menu is used in the prototype. The hypertext subsystem is geared to enable developers to explore "near matches", the second problem mentioned in section 3.3. The guided tours group together objects with similar functionalities. Developers can easily inspect similar objects by following these guided tours.

### 4.3. Illustration

The following example, based on our experience with the ICE toolset, illustrates our approach. The software repository under consideration contains various kinds of objects, as shown in Table 3.

| **Object Type** | **Description** |
|---|---|
| *Components* | Code in a third generation language, that is usually reused from other, non CASE based applications. |
| *Business Processes* | High level object that encompasses all other objects of an application. Supports a specific business activity. |
| *Rules* | Programs written in a fourth-generation programming language from which code is generated automatically and later compiled for the target platform. |
| *Views* | Prescription of interactions between the information stored in tables and the users as well as the interaction between various rules. |
| *Windows* | Descriptions -- in the form of templates -- of screens to be used as part of views. |
| *Fields* | The smallest unit of information, used to describe atomic information such as **customer name, product price**, etc. |
| *Tables* | Relations that are to be accessed using SQL. |

**TABLE 3: ICE OBJECTS**

Based on structured interviews with seven software developers conducted over a period of three months at three research sites, we were able to determine that reuse can be promoted by encouraging the reuse of *rule* objects, over other objects. *Rules* are written in a fourth generation programming language, which is high level. Figure 4 shows part of the code of a rule that performs an SQL operation. We discovered that developers' reuse of *rules* was less than one may have expected in view of the emphasis on reuse in the software development methods used. Although developers were apparently interested in reusing *rules*, they had

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

difficulties in locating those exhibiting appropriate functionally matches. Thus, we concluded that a tool to support reuse search for rules should result in significant benefits in reuse levels.

```
SQL ASIS
        DELETE FROM WP_PARTNER WHERE
                        WP_PARTNER_ID =
                        :WP_PARTNER_SQL_DEL_ID
ENDSQL
*> If a contact is deleted then you must also delete
   all information owned/related to him <*
```

**FIGURE 4: SAMPLE CODE OF AN ICE RULE**

We also discovered that software developers classify *rule* objects in terms of three major facets. The *repository structure* facet describes how a rule relates to other objects in the ICE repository. The *functionality* facet describes the kind of processing that the rule implements, e.g., calculations, database access. The third facet, *(3) business domain*, provides information about the application domain the rule deals with. This third facet normally describes a business entity such as customers, financial instruments, etc. Table 4 summarizes this classification schema[3] . A partial enumeration of several rules from ICE repositories is depicted in Table 5.

[3] The reader should recognize that this categorization is not exhaustive; it is merely illustrative. We currently have research underway that aims to elicit a more complete characterization of the classification schema that software developers use in this context.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

| **Repository** Classifier | Description |
|---|---|
| *DRIVER* | calls other *rules* |
| *INTERACTION* | uses other objects such as *views, windows* and other *rules* |
| *LEAF* | is not called by any *rule* |
| *ROOT* | is at the top of the calling hierarchy |
| *SUB-RULE* | is called by another *rule* |

| **Functionality** Classifier | Description |
|---|---|
| *SECURITY* | passwords, etc. |
| *CALCULATION* | numeric calculations |
| *CLIENT/SERVER* | communication protocols |
| *DIALOG* | interaction with user |
| *ERROR MESSAGER* | displays error messages |
| *EXCEPTION* | handles exceptions |
| *DISPLAY* | on screen display |
| *RETRIEVE* | Retrieve based on input criteria |
| *SQL* | database operations |
| *THREAD* | links objects |
| *SEARCH* | seeks and locates |
| *UTILITY* | general functionality, e.g., get date |
| *UTILITY* | general functionality, e.g., get date |

| **Domain** Classifier | Description |
|---|---|
| *ADDRESS* | e.g., customer addresses |
| *COMM_REC* | commercial requirements |
| *CONTACT* | contact person within a customer's firm |
| *CUSTOMER* | buyer firms |
| *FINANCIAL* | financial instruments |
| *FIRM* | general aspects of a company |
| *GENERAL* | applies to all domains |
| *PARTNER* | firm specifics |
| *PRODUCT MASTER* | financial and other instruments |

**TABLE 4: CLASSIFICATION SCHEMA FOR ICE RULES**

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

| Rule Name | Repository | Functionality | Domain |
|---|---|---|---|
| **ACCOUNT-LINK(#1)** | *INTERACTION* | *DISPLAY, SEARCH, THREAD* | *CUSTOMER FINANCIAL* |
| **ACCOUNT-LINK(#2)** | *INTERACTION* | *DISPLAY, SEARCH, THREAD* | *CUSTOMER FINANCIAL* |
| **APPROV-FRONT-CHK** | *DRIVER* | *SECURITY* | *CUSTOMER* |
| **COMMENT-DETAIL** | *DRIVER* | *DISPLAY, THREAD, UTILITY* | *CUSTOMER* |
| **CUST-INFO-UPDATE** | *INTERACTION* | *DISPLAY, RETRIEVE* | *CUSTOMER* |
| **CUST-PROD-1** | *INTERACTION* | *DISPLAY,* | *CUSTOMER* |
| **CUST-ACCOUNTS** | *LEAF* | *THREAD, SQL* | *CUSTOMER FINANCIAL* |
| **CUST-DELETE** | *SUB-RULE* | *DISPLAY* | *CUSTOMER* |
| **CUST-NAME-ADDR** | *LEAF* | *DISPLAY, SQL,* | *CUSTOMER* |
| **CUST_EXCPTN-DETL** | *SUB-RULE* | *DISPLAY, EXCEPTION* | *FINANCIAL CUSTOMER* |
| **CUST_EXCPTN-SUMRY** | *INTERACTION* | *DISPLAY, EXCEPTION* | *FINANCIAL CUSTOMER* |
| **DOC-TRACK-RETRIEVE** | *LEAF* | *SQL* | *FINANCIAL CUSTOMER* |
| **FIN1-INFO** | *LEAF* | *DISPLAY, SQL* | *CUSTOMER* |
| **FINU1-RETRIEVE** | *LEAF* | *DISPLAY, SQL* | *FINANCIAL* |
| **PRODUCT-DETAIL** | *SUB-RULE* | *DISPLAY* | *FINANCIAL* |
| **PMU909** | *LEAF* | *ERROR* | *PROD. MASTER* |
| **PMUCCC** | *DRIVER* | *SQL* | *PROD. MASTER* |
| **PMUXXX** | *DRIVER, SUB-RULE* | *SQL* | *PROD. MASTER* |
| **SALESMAN-NAME** | *INTERACTION* | *DISPLAY, SQL* | *FINANCIAL* |
| **SAVE-DATA** | *INTERACTION* | *DISPLAY, SQL, UTILITY* | *CUSTOMER* |
| **STORE-SQL-ERRORS** | *INTERACTION* | *ERROR, SQL* | *FINANCIAL CUSTOMER* |
| **SUBORDI_CUST_LINK** | *INTERACTION* | *DISPLAY, THREAD* | *FINANCIAL CUSTOMER* |
| **WP_CONTACT** | *DRIVER, INTERACTION* | *DISPLAY* | *CONTACT* |
| **WP_PARTNER_DYN_SQL_FET** | *SUB-RULE* | *SQL* | *PARTNER* |
| **WP_PARTNER_FIRM_SQL_FET** | *SUB-RULE* | *SQL* | *PARTNER* |
| **WP_PARTNER_SQL_SEL** | *SUB-RULE* | *SQL* | *PARTNER* |

**TABLE 5: SAMPLE CLASSIFICATION OF RULES IN A REPOSITORY**

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

The applications involved deal with investment accounts and with sales support systems. To illustrate the search process, let us suppose that the developer needs a high level rule to produce a report that provides information on the current status of all accounts for a given customer. To start the process of building the rule, the developer engages in the screening phase by issuing a request to retrieve all rules belonging to the *CUSTOMER* domain. The resulting sixteen rules are shown in Table 6.

| | Rule Name |
|---|---|
| 1 | ACCOUNT-LINK(#1) |
| 2 | ACCOUNT-LINK(#2) |
| 3 | APPROV-FRONT-CHK |
| 4 | COMMENT-DETAIL |
| 5 | CUST-INFO-UPDATE |
| 6 | CUST-PROD-1 |
| 7 | CUST-ACCOUNTS |
| 8 | CUST-DELETE |
| 9 | CUST-NAME-ADDR |
| 10 | CUST_EXCPTN-DETL |
| 11 | CUST_EXCPTN-SUMRY |
| 12 | DOC-TRACK-RETRIEVE |
| 13 | FIN1-INFO |
| 14 | SAVE-DATA |
| 15 | STORE-SQL-ERRORS |
| 16 | SUBORDI_CUST_LINK |

**TABLE 6: RESULTS FROM THE SCREENING PHASE**

The next step is identification. It involves the creation of hypertext guided tours linking the various rules. For each facet value that is shared by more than one rule, such a guided tour is created. The resulting eleven guided tours are shown in Table 7, where numbers refer to the rule numbers as they appear in Table 6.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

| **Repository Facet** | | |
|---|---|---|
| GT-1 | *INTERACTION rules* | {1,2,5,6,7,8,10,11,12,13,14,15,16} |
| GT-2 | *DRIVER rules* | {3,4,9} |
| GT-3 | *SQL rules* | {7,12,15} |
| | | |
| **Functionality Facet** | | |
| GT-4 | *DISPLAY rules* | {1,2,4,5,6,8,10,11,13,14,16} |
| GT-5 | *SEARCH rules* | {1,2} |
| GT-6 | *THREADING rules* | {1,2,7,16} |
| GT-7 | *UTILITY rules* | {4,14} |
| GT-8 | *RETRIEVAL rules* | {5,9,12} |
| GT-9 | *EXCEPTION rules* | {10,11} |
| | | |
| **Domain Facet** | | |
| GT-10 | *CUSTOMER* rules | {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16} |

**TABLE 7: THE HYPERTEXT GUIDED TOURS GENERATED BY USING THE FACETED CLASSIFICATION.**

Each of the guided tours enables a developer to explore similar rules, i.e., those that share the same classifier in a given facet. Figure 5 shows portions of guided tours GT-1, GT-3, GT-6 and GT-10. Since some of the guided tours intersect with each other, there are opportunities for a developer to move among exploration paths at intersection points. Figure 6 depicts a portion of the derived hypertext network that shows the intersection of guided tours at rule **CUST-ACCOUNTS** (rule 7). When a developer reaches that rule, she has the ability to continue exploring along any of the three guided tours, i.e. move directly to rule 8 (via GT-1 or GT-10), to rule 12 (via GT-3) or to rule 16 (via GT-6.)

As we see in Figure 6, rule 7 -- **CUST-ACCOUNTS** -- is related to rule 16 -- **SUBORDI_CUST_LINK** -- via **GT-4**, which groups *THREADING* rules. Rule 7 is connected to rule 8 via two links, representing the *INTERACTION* (**GT-1**) and *CUSTOMER* (**GT-10**) guided tours. Rule 7 is also connected to both rule 12 via the *SQL* rules guided tour and to rule 16 via the *THREADING* rules guided tour.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

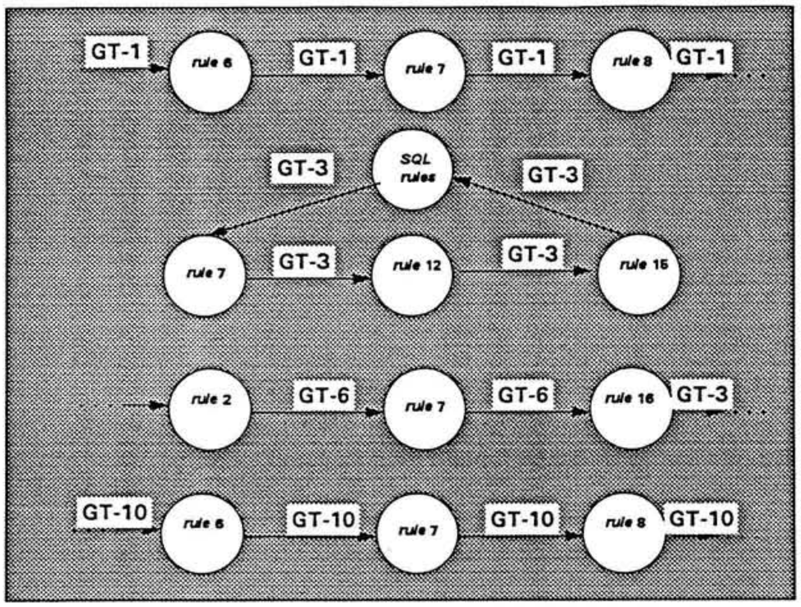


**FIGURE 5: THE FOUR GUIDED TOURS CONTAINING RULE 7**

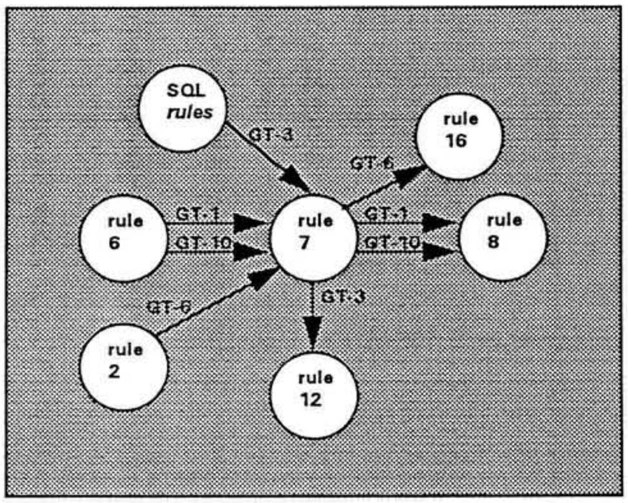
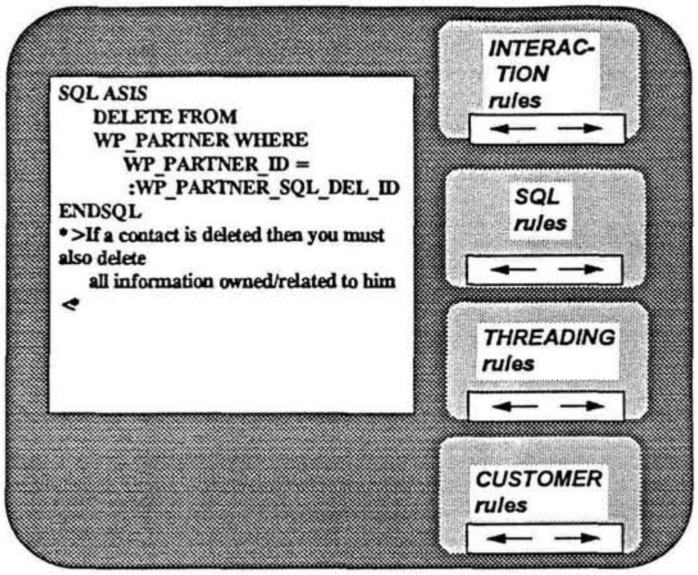


**FIGURE 6: A PORTION OF THE DERIVED HYPERTEXT NETWORK**

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47



```
SQL ASIS
    DELETE FROM
    WP_PARTNER WHERE
        WP_PARTNER_ID =
        :WP_PARTNER_SQL_DEL_ID
ENDSQL
*>If a contact is deleted then you must
also delete
    all information owned/related to him
<
```

**FIGURE 7: A SAMPLE SCREEN SHOWING A RULE AND THE NAVIGATIONAL LINKS EMANATING FROM IT**

Figure 7 illustrates a screen design for rule nodes. By clicking on the buttons, a developer can easily inspect various rules in searching for an object to reuse. By clicking on the left (right) arrow of any one of the highlighted criteria, the developer navigates to the previous (next) *rule* that guided tour. It is via these buttons that the hypertext system enables developers to navigate among software objects in a way that supports the search for reusable objects.

Our illustration demonstrates how to construct a hypertext network to support reuse from a classification of software objects. Moreover, it is clear that such construction can be automated. Since most of the information needed to classify ICE rules is present in the repository, the classification process may also be automated. Thus, we have shown how the principles of faceted classification and hypertext can be merged into an automated software tool to better support reusable object search. In section 5 we demonstrate the feasibility of our approach by presenting a prototype implementation of the tool to support search for reuse.

To address organizational aspects of software reuse, we point out that the tool for reusable object search is designed to be part of a larger set of reuse evaluation and support tools that represent the facilities of a commercial integrated CASE tool [2]. The other tools include a function point analyzer and an object reuse analyzer. The latter provides management with objective, inexpensively captured metrics that gauge the proportion of reused objects exhibited within and across repository applications. Once tools have been built

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

to support search for reusable objects, it will be possible to conduct experiments to determine their effectiveness and their impact on reuse levels.

## 5. IMPLEMENTATION: OBJECT REUSE CLASSIFICATION ANALYZER (ORCA)

We report on a prototype system for reuse search of repository rules that has been built within the ICE environment. The system is called ORCA, Object Reuse Classification Analyzer. In conformance with the search model presented in Section 4, the system consists of two components. The first component implements *screening* by combining an automated repository classifier and a query processor. This component enables system developers to specify repository queries based on the classification criteria. The classification is performed automatically by using information present within the ICE repository. Thus, developers do not have to manually classify the rules. The second component organizes the objects produced by the first component into a hypertext network and provides software developers with the functionality required to navigate among these objects.

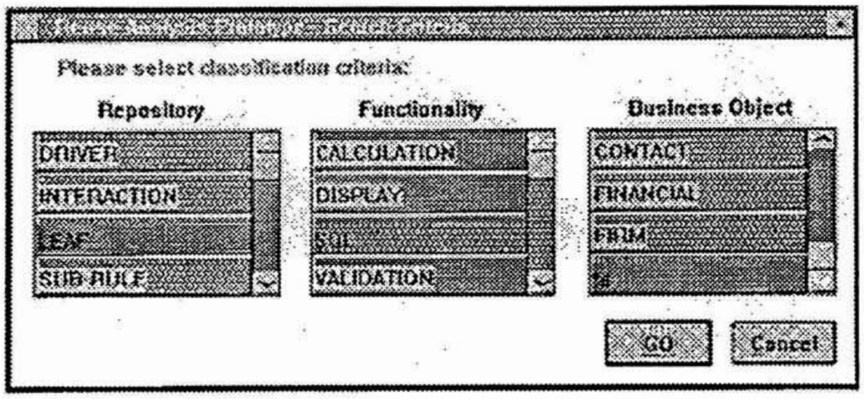
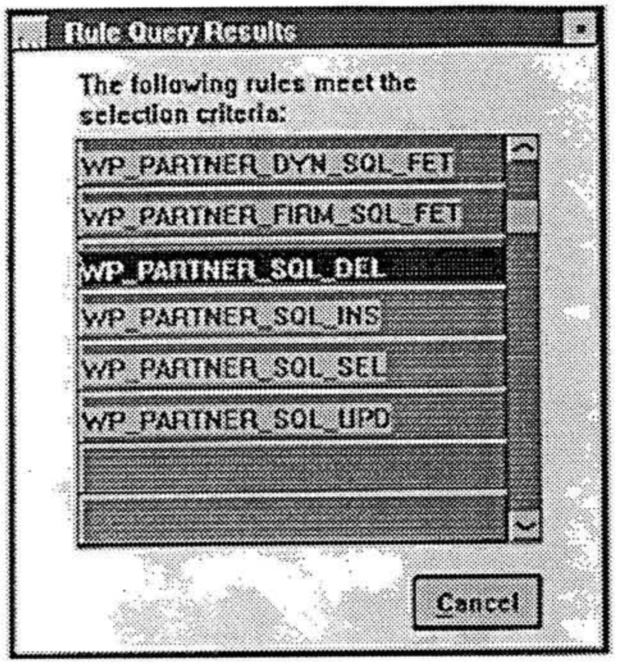


**FIGURE 8: THE SCREEN USED TO ENTER CRITERIA FOR SEARCH**

A sample session is depicted in Figures 8, 9 and 10. The system developer starts by selecting facet values from the three list-boxes[4] shown in Figure 8. Each list-box corresponds to one of the three classification criteria described in Section 4.5. In the figure, the user has specified a search for rules classified as *LEAF* and *SQL* in any business domain. The names of all the rules satisfying the criteria are retrieved into the screen shown in Figure 9. Developers select a rule by double clicking on its name. This brings up the detailed screen as shown in Figure 10. The classification boxes shown in the figure enable the user to focus on a more definite exploration. For example, by selecting *FIRM*, and clicking on the right (left) arrow, the developer is presented with the next (previous) *FIRM* rule from the list of Figure 9.

[4] The use of list-boxes ensures that developers choose valid classifiers, thereby eliminating word-choice problems.

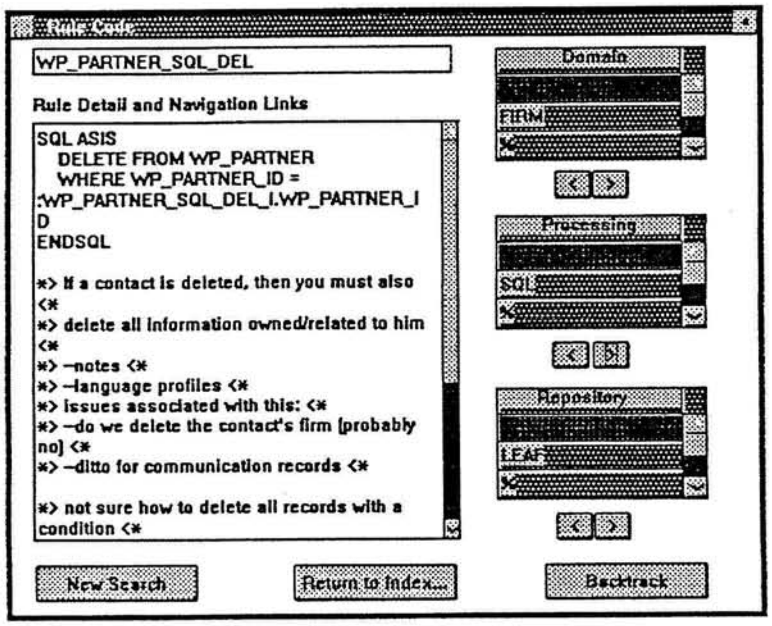
Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47



The following rules meet the selection criteria:

WP_PARTNER_DYN_SOL_FET
WP_PARTNER_FIRM_SOL_FET
WP_PARTNER_SOL_DEL
WP_PARTNER_SOL_INS
WP_PARTNER_SOL_SEL
WP_PARTNER_SOL_UPD

FIGURE 9: THE RULES SATISFYING THE SEARCH CRITERIA



WP_PARTNER_SQL_DEL

Rule Detail and Navigation Links

```
SQL ASIS
   DELETE FROM WP_PARTNER
   WHERE WP_PARTNER_ID =
:WP_PARTNER_SQL_DEL_I.WP_PARTNER_I
D
ENDSQL

*> If a contact is deleted, then you must also
<*
*> delete all information owned/related to him
<*
*> --notes <*
*> --language profiles <*
*> issues associated with this: <*
*> --do we delete the contact's firm (probably
no) <*
*> --ditto for communication records <*

*> not sure how to delete all records with a
condition <*
```

FIGURE 10: INSPECTING THE CONTENTS OF A RULE

***The left and right arrows enable hypertext navigation among rules satisfying the search criteria.***

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

At the core of the prototype system lies an algorithm that classifies objects based on information available in the repository. It does so in real-time. Each classifier is associated with a logic that prescribes when a rule falls within the scope of the classifier. When a developer selects criteria for search, the automated classification algorithm constructs a sequence of SQL queries by combining the logic associated with the each of the selected classifiers. The results are retrieved into a temporary area that is further processed to produce the hypertext network.

We identified two important implementation concerns through the development of the ORCA prototype. First, since criteria for classification are likely to evolve over time, the system should easily accommodate changes in the classification schema. In the prototype, criteria for classification are hard-coded. Hence, adding new classifiers requires re-compilation of significant portions of the prototype system. Second, fast response times are essential. Otherwise, developers will perceive the search costs to bee too high and will not use the search facility.

The construction of the prototype has helped us identify that classification speed and parameterization of classifiers are essential to successful implementation of a working system. Currently, ORCA has limited hypertext navigational facilities, and is restricted to rule objects and to a small set of classifiers. In spite of these restrictions, ORCA demonstrates the feasibility of the reuse search model presented in Section 4, as well as the feasibility of an automated classification algorithm without which a reuse search tool would be inefficient. The prototype serves as a proof of concept, and as a testbed to experiment with techniques and tools leading towards the design and construction of a working system.

## 6. CONCLUSION

In this paper, we have investigated the process of search for reusable software objects. Along with managerial and technical aspects of repository organization, the search mechanisms themselves are an important ingredient in determining the outcomes of programs that promote the reuse of software. We argued that the design of mechanisms to support search requires an understanding of the technical, cognitive and economic issues that arise in search:

1. From a *developers' technical perspective*, we noted that the search should be conducted at a level above that of source lines of code. At this higher level, a meta-language should facilitate the description of the software objects with attributes relevant to the developer [26]. Thus, we performed a series of structured

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

interviews with software developers that lead us to propose the classification schema presented in section 4.

2. From a *developers' cognitive perspective,* we suggested that developers need to overcome biases that develop out of their own experience and knowledge of a repository; without a means to "de-bias" their view of the repository, they can only fail to use it in the most efficacious manner. We have pointed out that search has a cost. According to the principle of equating marginal costs and marginal benefits; developers will engage in a search process only so as long as expected or perceived benefits outweigh the cumulative costs of search [34]. Technical tools to support search for reusable objects can substantially reduce these costs.

3. From a *firms' organizational perspective*, the approach we described leads to an integral environment to support software reuse. The search facility has been designed to match developers' own mental models. This will reduce the time required to train developers in the use of the search tool and will increase the likelihood that developers will use the search tool.

4. From a *firms' economical perspective*, we have designed the facility to support search for reuse so that it integrates well within existing repository-based CASE tools. This will reduce the cost of developing the search tool. The ORCA prototype, that was developed using the ICE tool, testifies to the feasibility of a rapid and inexpensive construction.

## 6.1. Contributions of the Research

Collectively, our review of these perspectives suggested the need for mechanisms to support reusable object search that can assist in:

1) interpreting the functionality of the objects being sought;

2) determining the qualities that are required for an object match;

3) searching the space of objects to produce candidates for reuse; and,

4) inspecting and evaluating those candidates.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

We propose a two-stage descriptive model of search for reusable software objects, that takes into account the three perspectives. During the *screening stage*, the user specifies a set of requirements to be satisfied. The screening tool exhaustively scans the repository seeking objects that satisfy these requirements. The outcome of this first phase is a set of potential candidates for reuse. During the *identification* phase, the user carefully scrutinizes this pool of candidates to select software objects for reuse. The process can be iterated: the developer refines the requirements specified at the outset until the desired objects are found, or until she terminates the search.

We conducted structured interviews with developers to define the specifics of a workable classification schema for ICE. This led us to establish a set of organizing principles for the object repository that are based on the mental models of the developers themselves. Moreover, the classification criteria can be derived from information that is present in the repository. As the prototype system demonstrates, this enables the automation of classification for reuse.

We argued that *a tool for search should support both phases of the process in a balanced manner.* After analyzing the requirements of each phase, we concluded that screening is better supported by a structured repository organization and a powerful query language. But *identification requires more active human participation*. Therefore, a different kind of tool -- for example, one that is based on *hypertext navigation of repository objects* -- is needed for identification. We have described the basic elements of a design for tools to support reusable object search and have given a proof of concept via the prototype implementation.

In order to validate and gauge the business value of the reuse mechanism that we have described in practice, we are planning to conduct a controlled experiment that involves novice developers in the software development consulting training program at the research site. They will be trained in the use of the CASE tool and the benefits of software reuse, and given a cursory overview of a large repository of objects available for reuse.

## 6.2. Caveats and Implementation Concerns

We now consider several caveats and implementation concerns that we have recognized in the process of carrying out this research.

*The Need to Accommodate Changing Classification Schema.* The reusable object search tools described in this paper will rely upon a classification scheme for software objects that we

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

expect will change over time. This change will occur as adjustments are made based on our experimental evaluation of how well the tool is performing, and developers' feedback and experience with it. A mechanism to support reusable object search that is inflexibly dependent upon a specific classification schema is quite vulnerable to changes in the schema, and therefore, highly undesirable. The reader should note that the query mechanism that is used during screening and the generated hypertext network reflect the classification schema. For this reason, we are taking extra care to develop a design that will enable the users to parametrize the classification schema, thus making it able to incorporate schema changes.

*The Need for Fast Response Times and Up-to-date Information.* The automatic classification and the construction of the hypertext network are quite time consuming since they require numerous queries to the repository. This discourages a dynamic system that classifies and builds in real-time. On the other hand, a batch process would be unable to incorporate recently created objects. Hence, developers might find a batch process less valuable. A good solution should exhibit an appropriate balance between speed and completeness of information.

*To What Extent Can Object Classification Actually Be Automated?* Since developers have little incentive to add keywords or other information conducive to assisting proper future classification, it is desirable that most of the classification process should be automated. Although it is conceivable that this may be achieved by inspecting software objects and their relationship with other objects in the repository, we should point out that the extent to which this is possible in practice (for the CASE tool at our research site or elsewhere) is really an empirical question. It may be hard, in fact, to come up with a fool-proof algorithm which will do so in a manner that requires no manual classification to complete the effort.

*Getting the User Interface "Right" Will Be Crucial.* As has been already documented elsewhere, e.g., [11], hypertext facilities that are embedded into other applications often fail because users are not accustomed to the "extra" interface. Our design for the hypertext aspects of the facility to support reusable object search has been chosen so that it blends well with the already familiar user interface of the CASE tool. This should ease acceptance and promote its use.

The technical solution for searching for reusable objects we proposed here is to be viewed in the context of a firm's efforts towards an integrated environment that supports reuse during software development and maintenance. From a managerial perspective, the technical tool will enable managers to set higher goals for reuse levels while providing developers with the means

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

to achieve these goals. Hence, our work will help promote higher productivity by combining technical and managerial aspects of software development.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

*: Affiliations of authors:
Information Systems Department
Stern School of Business
New York University

## REFERENCES

[1] U. Apte, C. S. Sankar, M. Thakur and J. Turner, "Reusability strategy for development of information systems: Implementation experience of a bank. *MIS Quarterly*, vol. 14, no. 4, pp. 421-431, December 1990.

[2] R. D. Banker, T. Isakowitz, R. J. Kauffman, R. Kumar and D. Zweig, "Tools for managing repository objects", in *Analytical methods for software engineering economics II*, P. T. Geriner, T. Gulledge and W. P. Hutzler, Eds. New York: Springer-Verlag, 1993.

[3] R. D. Banker and R. J. Kauffman, "Reuse and productivity: An empirical study of integrated computer-aided software engineering (ICASE) technology at the First Boston Corporation," *MIS Quarterly*, vol. 15, no. 3, pp. 375-401, September 1991.

[4] R. D. Banker and R. J. Kauffman, " Measuring the development performance of integrated computer-aided software engineering (ICASE): A synthesis of field study results from the First Boston Corporation," in *Analytical methods for software engineering economics I*, T. Gulledge and W. Hultgren, Eds. New York, NY: Springer-Verlag Publishers, 1993.

[5] R. D. Banker, R. J. Kauffman, and D. Zweig, "Repository evaluation of software reuse", *IEEE Trans. Software Eng.*, vol. 19, no. 4, pp. 379-389, April 1993.

[6] M. J. Bates, "Subject access in on-line catalogs: A design model," *J. Amer. Soc. Inf. Sci.*, vol. 37, no. 6 pp. 357-376, November 1986.

[7] B. Beckman, W. Van Snyder, S. Shen, J. Jupin, L. Van Warren, B. Boyd and R.Tausworthe, "ESC: A hypermedia encyclopedia of reusable sofware components," Jet Propulsion Lab., Calif. Inst. Tech., Pasadena, CA, September 1991.

[8] J. Bigelow, "Hypertext and CASE," *IEEE Software*, vol. 5, no. 2, 23-27, March 1988.

[9] J. Bigelow and V. Riley, "Manipulating source code in DynamicDesign," *Hypertext'87 Proceedings*, November1987, Chapel Hill, NC, ACM Press, pp. 397-408.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

[10] D. C. Blair and M. E. Maron, "An evaluation of retrieval effectiveness for a full-text document-retrieval system," *Commun. ACM,* vol. 28, no. 3, pp. 289-299, March 1985.

[11] M. L. Creech, D. F. Freeze and M. L. Griss, "Using hypertext in selecting reusable software components," *Hypertext '91 Proceedings,* ACM Press, San Antonio, TX, ACM Press, December 1991, pp. 25-38.

[12] J. Conklin, "Hypertext: An introduction and survey," *IEEE Computer,* vol. 20, no. 9, pp. 17-41, September 1987.

[13] J. Conklin and L. M. Begeman, "gIBIS: A hypertext tool for exploratory policy discussion," *ACM Trans. Office Info. Sys.*, vol. 6, no. 4, pp. 303-331, October 1988.

[14] J. L. Cybulski and K. Reed, "A hypertext-based software engineering environment", *IEEE Software*, vol. 9, no. 2, pp.62-68, March 92.

[15] R. Fidel, "Individual variability in on-line searching behavior," *Proc. Amer. Soc. Info. Sci. 48th Annual Meeting*, Las Vegas, NV. White Plains, NY: Knowledge Industry Publ., Inc., vol. 22, October 1985, pp. 69-72.

[16] G. Fischer, "Cognitive view of reuse design," *IEEE Software*, vol. 4, no. 4, pp. 60- 72, July 1987.

[17] G. W. Furnas, T. K. Landauer, L. M. Gomez and S. T. Dumais, "The vocabulary problem in human-system communications," *Commun. ACM,* vol. 30, no. 11, pp. 964-971, November 1987.

[18] P. K.Garg and W. Scacchi, "Ishys: Designing an intelligent software hypertext system," *IEEE Expert,* Vol . 4, no. 3, pp. 52-62, Fall 1989.

[19] P. K.Garg and W. Scacchi, "A hypertext system for software life cycle documents," *IEEE Software,* vol. 7, no. 3, pp. 90-98, May 1990.

[20] F. Garzotto, P. Paolini and L. Mainetti, "Navigation in hypermedia applications: Modeling and semantics, *J. Org. Comp.*, submitted for publication, 1993.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

[21] F. G. Halasz, "Reflections on Notecards: Seven issues for the next generation of hypermedia systems," *Commun. ACM*, vol. 31, no. 7, pp. 836-852, July 1988.

[22] T. Isakowitz, "Hypermedia in organizations and information systems: A research agenda," in *Proc. 26th Hawaii Intl. Conf. Sys. Sci.*, Maui, HI, vol. 3, January 1993, pp. 361-369.

[23] J. Karimi, "An asset-based systems development approach to software reusability," *MIS Quarterly*, vol. 14, no. 2, pp. 179-198, June 1990.

[24] P. Kerola and H. Oinas-Kukkonen, "Hypertext system as an intermediary agent in CASE environments," in *The Impact of Computer Supported Technologies on Information Systems Development,* K. E. Kendall, K. Lyytinen and J. DeGross, Eds. New York: North-Holland, pp. 289-313, 1992.

[25] Y. Kim and E. Stohr, "Software reuse: Issues and research directions," in *Proc. 25th Hawaii Intl. Conf. Sys. Sci.*, Maui, HI, IEEE Comput. Soc. Press, January 1992, pp. 612-623.

[26] Y. Matsumoto, "Some experiences in promoting reusable software: Presentation in higher abstract levels," *IEEE Trans. Software Eng.*, vol. 10, no. 5, pp. 502-512, September 1984.

[27] J. Nielsen, J. Hypertext and Hypermedia. New York, NY: Academic Press, 1990.

[28] J. Nielsen, "Navigating through hypertext," *Commun. ACM,* vol. 33, no. 3, pp. 297-310, March 1990.

[29] R. Prieto-Diaz, "Implementing faceted classification for software reuse", *Commun. ACM*, vol. 34, no. 5, pp. 89-97, May 1991.

[30] R. Robson, "Using hypertext to locate reusable objects," in *Proceedings of the 25th Hawaiian International Conference on System Sciences,* vol. 3, pp. 549-557, January 1992.

[31] D. Tarr and H. Borko, "Factors influencing inter-indexer consistency," in *Proc. Amer. Soc. Info. Sci. 37th Annual Meeting,* Atlanta, GA. White Plains, NY: Knowledge Industry Publ., Inc., pp. 50-55, October 1974.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47

[32] Tech. Comm. on Software Eng. of the IEEE Comput. Soc., "Working Draft on Standards for Reuse Measurements," IEEE Comput. Soc. Press, 1992.

[33] R. H. Trigg, "A network-based approach to text handling for the online scientific community," *Comput. Sci. Tech. Rep. no. TR-1346,* Dep. Comput. Sci., Univ. of MD, College Park, 1983.

[34] W. Van Snyder, "Sofware classification and retrieval," *Technical suuport package for NASA Tech. Brief NPO-18530,* NASA Tech. Briefs 17, 8, Item 27, August 1993.

[35] S. N. Woodfield, D. W. Embley and D. T. Scott, "Can programmers reuse software?" *IEEE Software*, vol. 4 no. 4, pp. 52-59, July 1987.

[36] W. A. Woods, "What's in a link: Foundations for semantic networks", in *Representation and understanding: Studuies in cognitive science*, D. G. Bobrow and A. Collins , Eds., New York" Academic Press, 1975 , pp. 82.

[37] P. Zunde and M. E. Dexter, "Indexing consistency and quality," *Amer. Documentation,* vol. 20, no. 3, pp. 259-264, July 1969.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-93-47