6-1996

Supporting search for reusable software objects

T. ISAKOWITZ

Robert J. Kauffman
Singapore Management University, rkauffman@smu.edu.sg

DOI: https://doi.org/10.1109/32.508314

Published in IEEE Transactions on Software Engineering, 1996 June, Volume 22, Issue 6, Pages 407-423
https://doi.org/10.1109/32.508314

Supporting Search for Reusable Software Objects

Tomás Isakowitz and Robert J. Kauffman

Abstract—Prior research has shown that achieving high levels of software reuse in the presence of repository and object-based computer-aided software engineering (**CASE**) development methods presents interesting human, managerial and technical challenges. This article presents research that seeks to enhanced software development performance through reuse. We propose automated support for developers who search large repositories for the appropriate reusable software objects. We characterize search for repository objects in terms of a multistage model involving screening, identification, and the subsequent choice between new object construction or reusable object implementation. We propose automated support tools, including **ORCA**, a software *Object Reuse Classification Analyzer*, and **AMHYRST**, an *Automated HYpertext-based Reuse Search Tool*, that are based on this model. ORCA utilizes a faceted classification approach that can be implemented using hypertext. We also describe an aspect of AMHYRST's architecture which can automatically create hypertext networks that represent and link objects in terms of a number of distinguishing features. We illustrate our approach with an example drawn from a real world object repository.

Index Terms—Classification, CASE, computer-aided software engineering, development environments, hypertext, object repositories, object search, repository evaluation, reuse, software development.

1 Introduction

SOFTWARE development methodologies that emphasize reuse are increasingly recognized by senior management in terms of the value they deliver in helping firms achieve higher levels of software development productivity and reduced software costs [1], [2], [21], [23]. Although software reuse is unlikely, by itself, to forestall the software development crisis, the recent attention that it has received is warranted. If firms are able to reduce the proportion of new code that must be constructed from 70% to 100% of the total, as in traditionally developed applications, to between just 30% and 40%—as we personally have observed in software development projects using CASE—the process of software development will improve significantly. To accomplish this, however, capital investment in tools that promote software reuse must occur.

A key ingredient for promoting software reuse in repository-based CASE environments is providing support for software developers who wish to search the repository to locate suitable software objects for reuse. This article explores conceptual and architectural bases for specifying a repository search tool which automates the process of repository search for a software object that is appropriate for a developer to reuse in a given situation. The tools that we propose combine two different capabilities: **ORCA**, the *Object Reuse Classification Analyzer*, and **AMHYRST**, the *Automated HYpertext Reuse Search Tool*.

• *T. Isakowitz is with the Department of Information Systems, Stern School of Business, New York University, 44 West Fourth Street, New York, NY 10012. E-mail: tisakowi@stern.nyu.edu.*
• *R.J. Kauffman is with Information Systems and Decision Sciences, Carlson School of Management, University of Minnesota, 271 19th Avenue South, Minneapolis, MN 55455. E-mail: rkauffman@csom.umn.edu.*

Manuscript received Nov. 15, 1993; accepted Apr. 22, 1996.
Recommended for acceptance by D. Perry.
For information on obtaining reprints of this article, please send e-mail to: transse@computer.org, and reference IEEECS Log Number S95417.

The conceptual basis of this work is a descriptive model that represents how software developers search a repository to find reusable software. From this perspective, classification and search represent activities that software developers currently perform without automated support. Classification approaches to promote software reuse have been proposed in earlier work, (for example, see [26]), however, there remains the need for additional research to address the challenges of CASE development. The architectural basis of our solution takes advantage of recent developments in hypertext technology. Hypertext applies well to domains where relationships among domain elements are important [20]. This is the case with software engineering, especially in CASE environments, where software artifacts, such as code, documentation and designs, are structured. (For example, programs call one another, programs use files, files and programs have documentation, etc.) Thus, when developers search for reusable software in a CASE environment they do so over a repository whose elements are related according to formal guidelines, for example, through a repository metamodel. This makes a hypertext-based solution suitable.

The meaning of the word *object* in the repository-based integrated CASE environment that we examine in this paper differs from the meaning it takes on in object-oriented environments, such as SMALLTALK or C++. In our context, the term *object* is used to denote elements of the software repository that are predefined in terms of the general functionality that they can provide. There are a limited number of object types, and they tie in closely with the manner in which the CASE tool enables software applications to be developed. In an object-oriented environment, an *object* represents a domain entity encapsulating data and functionality. As such, we follow Booch [9] in referring to our research environment as an *object-based software development environment*.

This article is organized as follows. Section 2 reviews background literature that provides motivation and evaluative guidance for the alternative approaches that are available to support developer search for reusable repository objects. Section 3 describes the *Integrated CASE Environment* (**ICE**) that is the testbed for our research. It also describes prior research conducted in this environment that explains why effective reuse search support is a precondition for further improvements. Section 4 introduces a multistage conceptual model of the search process for reusable objects. The primary argument is that search involves different activities: identification of potentially reusable objects and functionality screening. A related argument is that each activity needs to be supported in a different manner to maximize effectiveness. Then, in Sections 5 and 6, drawing on what we learned in structured interviews with ICE developers at two large firms, we show how a combination of object classification and search mechanisms can provide improved support for reusable object search. We illustrate these concepts with a realistic example that includes ICE objects similar to those that might be included in a customer service application at the research sites. The paper concludes with a discussion of a prototype implementation of the proposed tools.

2 Alternative Approaches to Support Repository Search for Reusable Objects

In this section, we begin by contrasting managerial and technical approaches to supporting repository search. We then evaluate the strengths and weaknesses of alternative technical approaches as a basis for crafting automated repository search support tools. A study of representation methods for software components conducted by Frakes and Pole [13] showed that there are no significant differences in search effectiveness among the methods, and that none of them provides adequate support for understanding the software objects to be reused. Two of the four methods covered in the study, faceted classification and keywords, are also discussed. Based on our observations and on Frakes and Pole's findings, we conclude that hypertext-based approaches offer a number of conceptual and technical features that are well-matched to the problem domain we are investigating.

2.1 Managerial versus Technical Support Approaches

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 project managers about the contents of the repository so that they can plan software applications to maximize reuse. A related approach is to appoint a person to manage the repository. Similar to the role of a database administrator with respect to data definitions and data quality, the role of a repository administrator involves screening objects to be stored in the repository to enforce quality, and defining the requirements for a set of widely reusable objects. A repository administrator can also act to minimize redundancy by preventing the addition of objects with overlapping functionality. These managerial approaches to search have been adopted with varying degrees of success.

By contrast, a *technical support approach* applies computerized tools to assist developers in identifying and retrieving objects that are suitable for reuse. An example of such work is Henninger's CodeFinder, which recognizes the difficulties that users often have in formulating queries for reusable repository objects [18]. In the absence of a powerful tool to support developers in their search for reusable repository objects, we expect that observed levels of software reuse will underperform management's expectations for a productivity payoff. What might be the basis of a tool that improves support for search? We next review four alternative techniques that have been considered in prior research for improving user support for a variety of search techniques: keyword search, full text retrieval, structured classification schemata, and hypertext.

2.2 Keyword Search

Keyword search requires assigning to each software object a number of relevant keywords or indices. As an example, consider a firm that has developed a number of in-house applications using a centralized repository. Within the general ledger application there is a 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. So, 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. In software development settings, keyword-based search would require developers to provide appropriate keywords for *every* object in the repository. However, we know well from prior research that software developers do not willingly assign keywords to the software objects they create: there is no perceived direct benefit for the extra level of effort involved. Another objection to the keyword method centers on the ambiguous nature of keywords; substantial disagreement over the choice of keywords can occur when different words mean different things to different people [5], [14]. Therefore, keyword search has been found to offer limited power or to be impractical in many kinds of applications.

2.3 Full-Text Retrieval

The high cost of manual indexing makes it attractive to automate the indexing process. The simplest kind of automatic indexing occurs in *full-text retrieval* systems. Such systems work on the basis of a mechanism such as the following:

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 collection, the computer is instructed to search for all documents containing certain words and word combinations, which the user has specified. ([8], p. 289)

Full-text search works best for software objects that have embedded or attached comments. Full-text retrieval systems preprocess stored data or documents and construct index tables ahead of time. Then, user search is effected through a table lookup, which is speedy and efficient. Speed, however, is not the only relevant criterion. For example, Blair and Maron [8] showed that for large textual bases, full-text retrieval misses many relevant objects—as many as 80% of them. One can imagine situations in which full-text retrieval systems return too much information, inundating the user with unusable data. As a result, this search method applied in software engineering contexts would represent a "brute force" approach to the problem, taking into account little of what is known about how full-text retrieval fails to deliver in other search settings.

2.4 Structured Classification Schemata

Structured classification schemata use a fixed number of predetermined perspectives, or facets, for classification. Table 1 below contains sample entries from a library of software routines using a six-facet classification schema due to Prieto-Diaz [26]. To search for a software routine, a developer issues a query consisting of a sextuple of values that is compared to values describing routines in the software library.

A common problem with this approach lies in mishandling synonyms and misinterpreting words that have some lexical ambiguity. Inadequate treatment of synonyms can result in the retrieval of objects that are irrelevant to the search. Related to synonyms is the problem of *near matches.* These occur when software components are retrieved that closely resemble, but do not exactly match the query. To solve this problem, Prieto-Diaz proposed the use of a *conceptual graph* that determines a "distance" between near matches and the desired object, in terms of their facet values. By assigning a number to represent this relative distance, one can then rank the relevance of objects to a particular query. Lexical ambiguity, on the other hand, can cause low retrieval rates when only a few possible meanings of a word are considered. It can also lead to the retrieval of irrelevant objects when unintended word meanings are considered. A way of addressing these issues is to limit the vocabulary for classifying software components, and to only allow queries drawn from this controlled vocabulary.

Although the faceted classification of Prieto-Diaz fits in well with 3GLs, it does not exploit the characteristics of CASE environments. CASE repositories contain a wider variety of software objects, as well as more detailed information about relationships among them than those considered in non-CASE software libraries. For example, Prieto-Diaz's classification does not consider repository information. In the context of CASE development with Texas Instruments' IEF [19], for example, which includes many different kinds of objects that have different purposes, one would need to make distinctions that are finer that those contemplated in the classification schema shown in Table 1. In addition, the higher levels of abstraction that CASE tools enable, as compared to 3GL, render aspects of the Prieto-Diaz approach obsolete. In this vein, the facets **objects** and **medium** are unlikely to be relevant in many CASE environments because objects are represented independent of their implementation. However, as we will show in Section 5, it is possible to adapt faceted classification in view of the application metamodel that the CASE environment presents to a developer. (For additional information on metamodels as specialized database schemas in this kind of context, the interested reader should consult [29].)

Prieto-Diaz's faceted classification is also somewhat inflexible in another respect: It requires all objects to be classified in terms of the same facets. By contrast, Snyder [28] proposes a classification mechanism that allows for software objects to be classified along specialized facets without imposing them on all software objects. This approach, based on semantic networks [31], can also be adapted to CASE environments.

A disadvantage of structured classification approaches compared to full-text retrieval is that they require manual classification: automatically deducing software functionality is by no means a simple task. In integrated CASE environments, however, information available in the repository can be exploited to automatically classify software objects. Furthermore, the centralized software repository supports access to software objects for inspection purposes. The potential for automated classification and computerized support for access increases when the repository is organized in accordance with a metamodel, as is the case with many CASE environments. (Texas Instruments' *IEF* is again a good example.) Such environments provide the opportunity to automatically classify software objects and to support exploratory activities for software reuse.

2.5 Hypertext

Hypertext represents one of the newest forms of computer-based support for organizing 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 or other visual objects. The basic building blocks in hypertext are nodes and links [17]. Each *node* is associated with a unit of information, and nodes can be of different types. 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. (The interested reader should refer to Conklin [11] and Nielsen [24] for introductions to the capabilities of hypertext.)

Hypertext technology applies well to CASE environments because the information units (including software objects, documentation, among others) exhibit relationships among domain elements that are clearly defined. Hence, it

TABLE 1
FACETS IN THE CLASSIFICATION SCHEMA OF PRIETO-DIAZ [26]

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*

is appropriate to represent them as hypertext links. Also, the browsing capabilities of hypertext have the potential of supporting developer discovery about the contents of the repository. As developers become familiar with the repository contents through hypertextual navigation, one expects them to become more proficient in exploiting the available opportunities to reuse software objects. Current hypertext systems provide users with sophisticated user interface tools that enable them to inspect node contents, and to flexibly navigate through a network of nodes. For example, clicking on the name of a customer will result in a display of a detailed customer 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 module **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 [25], hypertext systems usually provide backtracking and other aids to navigation such as maps, to help orient the user.

A helpful hypertext concept—one that we employ later in this article—is that of a guided tour [16], [27]. In a guided tour, navigation is usually constrained to a few choices. An example would be when navigation can only proceed in two directions: either backwards or forwards through an ordered list of nodes. Although guided tours that are constrained in this manner would seem to limit the power of hypertext, they actually help reduce the disorientation that users experience when they confront a large number of navigational possibilities. For example, the collection of all software objects in a repository that implement a "customer SQL-update" can be organized into a guided tour. System developers seeking to implement SQL queries can navigate among the various elements of the guided tour to locate the ones that most closely match their needs.

Hypertext has been used previously to organize software repositories. For example, Garg and Scacchi's *Document Integration Facility* is a hypertext system that supports the development, use and maintenance of large-scale systems and their life cycle documents [15]. Bigelow and Riley's *Dynamic Design* is a hypertext-based repository that organizes relationships between various kinds of software, including specifications, design documentation, program documentation, user documentation, source code, object code and symbol tables [7]. Beckman et al.'s *ESC* project organizes software sources and documentation as a hypermedia encyclopedia to foster reuse. *ESC* can integrate distributed repositories by communicating with servers across a telecommunication network [6]. Creech, Freeze, and Griss describe *KIOSK*, a hypertext system to access a structured library of software components [10]. *KIOSK* supports multiple views of the library that correspond to the various roles of those involved in software development (e.g., developers, designers, users). Finally, Kerola and Oinas-Kukkonen have proposed the use of intelligent agents to facilitate interaction within a CASE environment through hypertext [22].

Except for *ESC* [6] and *KIOSK* [10] little research has been performed on the potential of hypertext to directly support search for reusable software objects. As Creech, Freeze, and Griss report [10], *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.

The structured nature of repositories in CASE environments also provides opportunities to incorporate hypertext facilities in applications developed with CASE. For example, Chen [12] discusses extending repository-based CASE environments used to construct executive information systems (**EIS**). The extension includes models of the information systems being developed and of the organizations that use them. The resulting EISs provide users, mainly business executives, with graph-based user-interfaces to guide their navigation in search for information.

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. Based on a navigational metaphor, hypertext represents a fourth approach that can be implemented in conjunction with a variety of methods to give users the ability to navigate the space of repository software objects at their own will. Hypertext also opens up the possibility for a developer to refine the repository query strategy to reflect an evolving understanding of how the query ought to be formulated [25]. In this article we present a proposal to combine automated classification and hypertext in a tool that provides navigational capabilities for repository objects. We now turn to a more detailed description of the software engineering environment that forms the testbed for our work.

3 ICE—AN INTEGRATED CASE ENVIRONMENT

The *Integrated CASE Environment* (**ICE**) is a computer-aided software engineering tool set enabling software developers to write applications using a fourth generation language, yet it buffers them from the complexities of the multiple languages and diverse hardware platforms encountered in modern client-server system development. Creation of this environment initially was undertaken at a large New York City-based investment banking firm which believed that it was mission-critical to refocus the firm's software development strategy to emphasize software reuse [2]. The intent was to speed software delivery and shorten the cycle time needed to put software support into place for new financial products. Since its initial deployment in the latter part of the 1980s, the capabilities of ICE have been expanded considerably. The tools have been commercialized and are now used in over 100 large firms around the world.

3.1 Definitional Considerations: Software Objects in ICE

When we refer to software objects in ICE, we recognize their basis in the *entity-relationship model*, rather than in the more general paradigm of object-oriented software. Application functionality derives from the interplay of repository objects organized into an application hierarchy. Each object can perform different functions, depending on the kind of object it is, and what a software developer programs it to do. Because ICE is also *repository-based*, software developers rely on the repository as the single store for all the pieces of software that make up the firm's applications.

The repository stores objects of different types in pre-specified templates that indicate their type. Table 2 provides definitions for the set of ICE repository objects. ICE object repositories can be organized with hypertext technology because each of the ICE object types relates to the other object types in a pre-specified and structured manner. Fig. 1 illustrates the relationships among them, and can be considered a metamodel for all application development in ICE. The only aspects of the metamodel that we do not depict are that multiple RULE SETS combine to form the software support for a BUSINESS PROCESS, and that a comprehensive set of repository BUSINESS PROCESS objects make up the entire inventory of CASE-developed applications of the firm.

The reader should think of the different object types in terms of corresponding 3GL constructs. RULE SETS, collections of individual ICE RULES, form application procedures. (Hereafter, we refer to RULE SETS simply as RULES.) RULES typically contain the instructions that software developers would associate with "the program." RULES use other RULES and COMPONENTS; RULES create REPORTS which consist of REPORT SECTIONS; and, RULES access DATA TABLES and interact with WINDOW DEFINITIONS, enabling an application developer to create a user interface. A REPORT can be thought of as a means of formatting data that result from queries on an application database. Fig. 2 presents an example of a simple ICE RULE that performs DATA TABLE deletion.

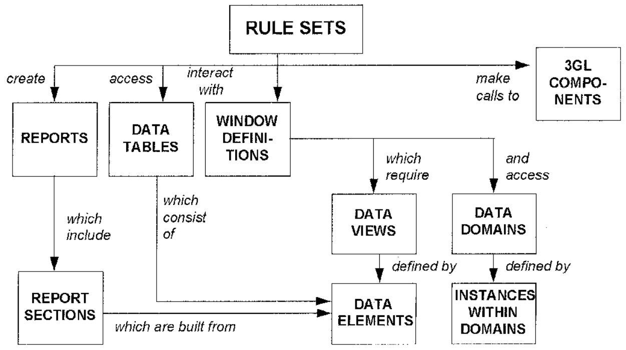


Fig. 1. The Repository Metamodel for ICE software applications. Note: 3GL COMPONENTS are available for use by RULES, however, they are not stored in the repository. All other object types are organized within ICE applications according to the structure described in this figure. The reader should recognize the well-defined relationships and syntax. These features enable us to navigate ICE repositories with the node and link paradigm that characterizes hypertext.

```
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 that contact<*
```

Fig. 2. Sample 4GL code for an ICE RULE.

TABLE 2
REPOSITORY OBJECTS IN THE INTEGRATED CASE ENVIRONMENT

Object Type	Description
BUSINESS PROCESSES	High level objects that act as the top of the hierarchy of an ICE application and that define the software support available for a specific business process or sub-process.
RULE SETS	Procedures written in a 4GL language supported by ICE that perform most of the traffic control and processing associated with ICE application software.
3GL COMPONENTS	Reusable modules of 3GL code that are stored outside the ICE repository, but are callable from within it.
REPORTS and REPORT SECTIONS	A repository representation of a physical report, including application output. REPORTS consist of multiple REPORT SECTIONS, each with its own data requirements and format.
DATA TABLES	Relations that can be accessed using SQL queries.
VIEWS	Objects that mediate the interactions that occur among RULE SET and DATA ELEMENT objects, especially when data stored in TABLES are required to populate a user WINDOW.
WINDOW DEFINITIONS	A logical definition, stored as a template, of an on-screen image in the form of a window.
DATA DOMAINS and INSTANCES WITHIN DATA DOMAINS	These ICE objects work together to specify the range of values that DATA ELEMENTS can take on when they are delivered to WINDOWS and DATA TABLES.
DATA ELEMENTS	The smallest unit describing data items, such as customer name or product price, that are used to define DATA TABLES.

3.2 Observations About Software Reuse Related to ICE Development

Software reuse in ICE is accomplished when an application object makes a call to an object that already exists in the repository. Thus, reuse results when an object that was developed specifically for an application is called multiple times. Another instance of reuse occurs when a developer designs an ICE application to make a call to an object that was developed in the context of another ICE application, and, as a result, has been stored in the repository. Although ICE supports other kinds of reuse, for example, during the design, testing, and production phases of the software development life cycle, our research focuses exclusively on reuse that occurs during software construction, where software developers experience the greatest difficulties in finding reusable software objects.

Experimental development of a number of small, but realistic repository object-based ICE applications indicated that two of every three application object calls were delivered through reuse [2], [3]. Later, in large-scale software development projects at an investment bank and at a large national retailer, this level of object reuse was often exceeded. It is especially 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 especially powerful tools in place to encourage reuse. In fact, at the time, we speculated that the observed reuse levels were a conservative estimate of what could actually be achieved if additional technical support features and new developer incentives were implemented in the presence of a mature repository [4].

However, reuse levels did not exhibit significant increases even as new applications were developed and developers became better at using the CASE tool. Anecdotal evidence that was obtained as we debriefed project managers on our results pointed to weaknesses in the support mechanism that inhibited less experienced developers from identifying opportunities to reuse existing repository software objects, especially those created in other projects. For instance, we learned that 60% of software reuse involved objects written and reused by the same developer, and 85%-90% of software reuse involved objects constructed by members of a project team within the same application.

Corroborating evidence in a different development environment was obtained by Woodfield, Embley, and Scott [30], who examined the performance of programmers that were relatively untrained in reuse. Although they only looked at reuse of abstract data types stored in a software library, their results suggested that individual assessment of what elements are thought to be important in identifying targets for reuse will constrain performance. They also reported that if the effort to reuse was perceived to be less than 70% of the effort to build similar functionality, then an attempt would be made to actually reuse existing software; otherwise, new code would be constructed. These findings prompt us to consider the nature of a technical solution to improve developers' search for reusable repository objects as a means to improve their performance in software reuse.

4 A Multi-Stage Model for Reusable Object Search

A key determinant of success in promoting more software reuse is understanding the process that software developers engage in that leads to reuse. We conceptualize repository search as a set of related activities, each of which places somewhat different demands on developers with varied levels of knowledge of a repository's contents. This conceptual model is depicted in Fig. 3.

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. One would expect variation in software developers' ability to screen for relevant objects, based on a number of factors that constrain their knowledge of the repository. Such knowledge includes the developer's experience with ICE tools and the object repository, the overall skill level as a software engineer, application domain-specific experience, the maturity and contents of the repository, and individual efforts to learn about potentially reusable objects.

During Stage 2, *identification*, developers closely examine the subset of objects assembled during screening to determine if any of its objects provides the desired functionality. A number of factors are likely to influence a developer's performance in identifying objects with appropriate functionality. These include the nature of the software development environment and the extent to which it has been crafted to emphasize reuse, the repository or application metamodel or structure, the kind of application object that is sought (general or highly specific in function), the kind of satisficing metric that a software developer applies to determine if an object will deliver the necessary functionality, and the presence of a reusable object search support facility. Stage 3 is a *decision* phase which completes the process. A developer must decide whether to implement a reusable repository object, or scratch build a new one. We do not consider the decision phase in the present research.

The underlying idea of the model 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—the relevant objects like needles in a haystack—it is unlikely that a developer will be able to locate the requisite functionality in Stage 1. On the other hand, we would expect the developer to be more proactive in Stage 2, where identification occurs from among a smaller number of objects.

5 A Conceptual Basis for Reusable Object Classification

We conducted a set of structured interviews in field studies of ICE development practice at two large organizations, an investment bank and a specialized software development consultancy. The interviews were aimed at discovering a classification schema for ICE repository objects. In these interviews, we learned about key issues that a solution would need to address and the demands that would be placed on a classification mechanism for repository objects.

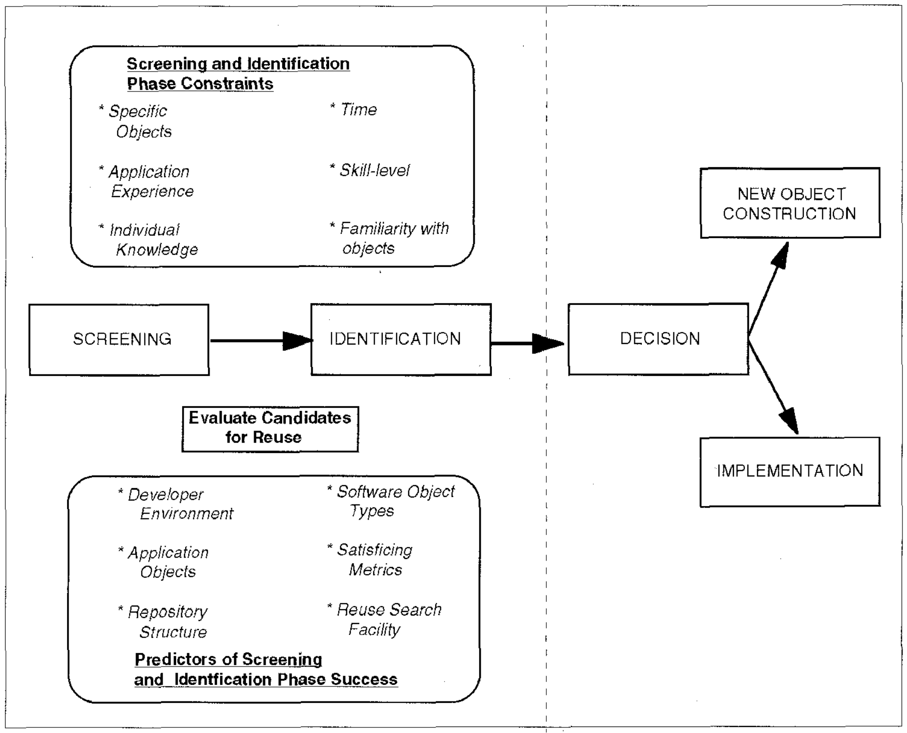


Fig. 3. A multistage descriptive model of search for reusable software objects.

5.1 Interviews with Software Developers

Preliminary Interviews. The purpose of the preliminary interviews was to establish a structured format for the substantive interviews that would enable us to specify a reusable object classification. Three software developers participated. These interviews reinforced what we learned in prior research about reuse of ICE repository objects: developers reported that reuse of RULES was less than what might be expected. They reported difficulties with the keyword-based repository search facility that was then available as a part of the ICE tool set. Developers also indicated that reuse of RULE objects, not surprisingly, would yield the highest payoffs by reducing overall costs in the construction phase. They suggested to us that a good first step in improving the support for repository search would be to pilot a facility that improved their ability to search for RULE objects. In response, we decided to limit the scope of our investigation in later interviews to the reuse of RULE objects only. As a result, the illustration that appears later in this article will be limited to ICE RULES only.

Interviews to Establish a Repository Object Classification. Seven developers participated in taped interviews of one hour or more. The subjects included two experts who had more than twelve years of general software engineering experience each, two novices with about one year of experience, and three people with five to eight years of experience. The interviews had two parts. The *closed-ended portion* of the interview attempted to elicit information about the kinds of information that would be necessary to exhaustively classify ICE repository objects. The classification task focused on repository RULES, and developers' responses suggested a strong awareness of the ICE metamodel. We asked them to choose 10 objects from a repository that they knew well, and develop a classification scheme with as many as eight dimensions. They were also asked to estimate the extent to which these objects were reused. The *open-ended portion* of the interview was intended to reveal issues that the developer thought were important, either to clarify responses to the closed-ended questions or to explore other related issues that might have a bearing on the feasibility of specifying a workable classification mechanism.

Developer exposure to the ICE environment varied, with less than one year at the low end and approximately four years (the age of the tool set at that time) at the high end. Participants also were asked to estimate how many objects they were familiar with in their development environments. Expert developers indicated a knowledge of or experience with between 5,000 and 10,000 repository objects based on three to four years experience with ICE. Novices knew much less about the repository, reporting familiarity with an average of about 300 objects. Other developers indicated knowledge of between 1,000 and 3,000 objects, with an average of about 1,500. Awareness of object reuse levels varied, with experts indicating more awareness of the potential for a given object to be reused.

5.2 A Classification Schema for RULE Objects in an ICE Repository

The most important finding in the interviews was that it was possible to construct a robust classification schema for

ICE repository objects. ICE software developers classified RULE objects in terms of the three facets: *repository, functionality* and *domain*. Table 3 presents this classification schema.[1] This general approach is applicable to other ICE objects, and, in principle, to other repository-based CASE environments, too.

The Repository Facet. This describes how a RULE relates to other objects. For example, a RULE may call or be called by another RULE, and is classified as a *DRIVER* or a *SUB-RULE*, as a result. A RULE that enables user interaction via a WINDOW is classified as an *INTERACTION*. It also is possible for a RULE to be classified under many categories. For example, referring to Table 3, a RULE can be a *ROOT* and a *DRIVER*, a *SUB-RULE*, and a *DRIVER*, or a *DRIVER* and an *INTERACTION*—all at the same time. We believe that the repository facet emerged as a result of developers' awareness of the repository object metamodel. Developers work within the context of this metamodel, so their mental model reflects the repository structure. Hence, they classify RULES in terms of their relationship to other repository objects.

The Functionality Facet. This facet describes the kind of processing that the RULE implements, e.g., calculations, database access, security, etc. It is possible to further refine the classification shown in Table 3, e.g., by specifying the kind of *SQL* operation in terms of a "CRUD" matrix, involving actions to create, retrieve, update, or delete. This facet also occurs in other CASE environments. For example, in Texas Instruments' *IEF* [19], Process Action Diagrams (**PADs**) provide the processing logic for an elementary process that always has at least one CRUD operation against one or more entity types. Common subroutines are called Process Action Blocks (**PABs**), and Procedure Action Diagrams (**PRADs**) start interaction screens. Thus PAD corresponds to *SQL*, PAD to *SUB-RULE*, and PRAD to *INTERACTION*.

The Business Domain Facet. We also learned that developers classify software objects in terms of the general domain in which the objects are used. In ICE, as in other CASE environments, software objects form part of larger applications that support operations in specific business domains. Examples of such applications involve customer support, salesperson contacts, and pretrade analysis of financial instruments. The corresponding business domains are *CUSTOMER, CONTACT*, and *PRODUCT MASTER*. The business domain facet of a RULE records the domain of the application to which the rule belongs. However, if a RULE is reused in more than one application, there will be multiple entries.

5.3 Classification Illustration

To illustrate these findings, we consider the schema's ability to classify members of a set of ICE RULES that might be encountered in a prototypical customer account application, as shown in Table 4. Table 5 shows how these RULES would be classified in terms of the **repository**, **functionality** and **domain** facets.

1. 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.

Some explanation regarding the values for the functionality facet may help readers understand the example better. The classifiers in the table are DISPLAY, ERROR, SEARCH, SECURITY, SQL, THREAD, and UTILITY. DISPLAY RULES present information to and elicit information from users. By definition, a RULE is a DISPLAY RULE if it *interacts* with a window. ERROR RULES perform error-checking routines, such as displaying an error message. SEARCH RULES perform select queries on a database. SECURITY RULES deal with password and other kinds of protection. SQL RULES, as the name indicates, perform SQL queries. Some more specialized SQL RULES have more definite classifiers, like SEARCH, already mentioned, and THREAD. THREAD RULES implement linked lists. For example, linking a customer record to all the customer's account activity records. Finally, UTILITY RULES perform general utilities that are used in multiple settings, such as transforming dates into text. Experienced developers understand these facet values and are readily able to apply them to describe the object. Novices obviously would need to be trained to understand how to distinguish among the facet values to adequately describe a repository object.

6 A Hypertext-Based Architecture to Support Reuse

Hypertext has the potential to provide a more powerful search capability than existing approaches offer. Our goal in this section is to illustrate how hypertext can be brought to bear on the different problems that developers face in Stages 1 and 2 of the reusable object search process.

6.1 Stage 1—Support for Screening

Using our classification schema, the user would specify the functionality to be implemented via an interactive screen in which descriptors, belonging to the various facets, are elicited. Screening consists of retrieving from the repository a set of objects that belong to the classification specified by the developer. The objective during this stage is the retrieval of a manageable set of candidates for reuse. By the end of screening, a sizable set of potentially relevant objects—the set of candidate objects for reuse—will have been extracted from the repository. Some of the retrieved objects may not be completely relevant to the task to be implemented, but most of the relevant ones ought to be included. It would be too labor-intensive to individually examine each of them at this point: there will still be too many. Instead, the candidate objects need to be organized to facilitate inspection during the identification stage.

Next, the set of candidate objects that is obtained can be structured as a network of hypertext guided tours according to the classification schema. Although the creation of the hypertext guided tours is transparent to developers, they will use it in Stage 2 to inspect objects for reuse. Fig. 4 illustrates the use of guided tours to interweave related software objects.

The objects shown in the figure all have the same value f for facet F. Construction of the guided tours is based upon the following algorithm:

TABLE 3
A CLASSIFICATION SCHEMA FOR ICE RULES

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 MES-SAGER	displays error messages
EXCEPTION	handles exceptions
DISPLAY	on screen display
RETRIEVE	retrieve based on input crite-ria
SQL	database operations
THREAD	links objects
SEARCH	seeks and locates
UTILITY	general functionality, e.g., get date

Domain Classifier	Description
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
ILLUSTRATION: SAMPLE RULES IN AN ICE OBJECT REPOSITORY

RULE Name	Description
ACCOUNT-LINK(#1)	*Links and displays all accounts available for a given customer.*
ACCOUNT-LINK(#2)	*Displays all accounts for a given customer given the account name or account number*
APPROV-FRONT-CHK	*Checks whether the ID entered by the user matches the terminal ID.*
COMMENT-DETAIL	*Lets user type in free-form text and stores it in the corresponding customer detail.*
CUST-ACCOUNTS	*Retrieves accounts linked to a customer.*
CUST-DELETE	*Begins logical delete process. SQL is performed by sibling rules, not here.*
CUST-INFO-UPDATE	*Retrieves customer information from the database and displays it in a window.*
CUST-NAME-ADDR	*Displays information on the main window and allows the user to input information and to change table values.*
CUST-PROD-1	*Displays a message containing product information.*
CUST_EXCPTN-DETL	*Displays detail for an exception.*
CUST_EXCPTN-SUMRY	*Displays exception window. Conditional.*
DOC-TRACK-RETRIEVE	*Retrieves document dates*
FIN1-INFO	*Retrieves and displays financial information for group #1*
FINU1-RETRIEVE	*Retrieves and displays financial window*
PMU909	*Common error handler for product manager*
PMUCCC	*Driver rule for product manager update / copy-add*
PMUXXX	*Driver rule for product manager add*
PRODUCT-DETAIL	*Display product detail*
SALESMAN-NAME	*Display salesman name listbox. Calls retrieval rule.*
SAVE-DATA	*Displays a message asking whether to save the data or not.*
STORE-SQL-ERRORS	*Processes SQL error, puts table, displays message*
SUBORDI_CUST_LINK	*Displays customer children in a listbox*
WP_CONTACT	*Calls other rules to display customer contact information*
WP_PARTNER_DYN_SQL_FET	*Populates a listbox based on input criteria.*
WP_PARTNER_FIRM_SQL_FET	*Fetch/Join WP_FIRM and WP_PARTNER*
WP_PARTNER_SQL_SEL	*Select from WP_Partner*

TABLE 5
RULE CLASSIFICATION: AN ILLUSTRATION

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*

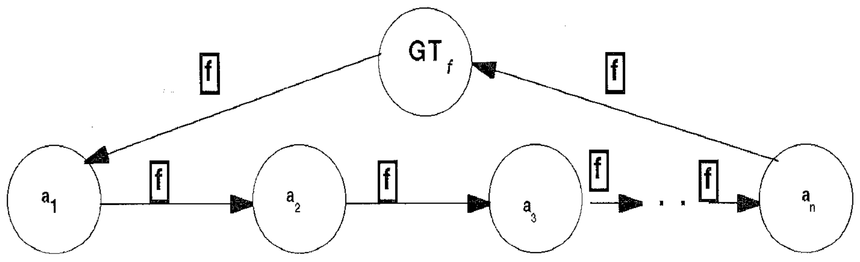


Fig. 4. A guided tour, GT_f, connecting objects a_1, a_2, ..., a_n with facet value f.

If a_1, a_2, ..., a_n are all the objects obtained from screening with a value of f for facet F, they are collected into a guided tour $GT_f = \{a_1, a_2, ..., a_n\}$. First the objects are arranged in lexicographic order by rule name.[2] Then links labeled f are created to connect a special start node labeled 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.

2. The ordering criteria can reflect more specific information, such as object similarity, if this is available.

6.2 Stage 2—Support for Identification

Now, a developer can proceed to explore the set of candidates during the identification step using a hypertext-based tool that enables inspection of various objects by navigating from object to object, within the set of candidates for reuse. The navigational capabilities of hypertext facilitate rapid traversal of the network to locate target objects for reuse. Navigation is helpful in zeroing in on the requisite functionality because the links are set up according to a classification schema that reflects the organization of the repository.

The network of interconnected links obtained from the screening phase gives developers additional capabilities to learn about related software functionality in the repository. Fig. 5 shows a portion of two intersecting guided tours, one linking RULES classified under the facet value ***INTERACTION*** (including RULES 7, 8, 10, 11, and 12), and the other linking RULES classified as ***SQL*** (RULES 7, 12, and 15).

It is possible to use techniques other than hypertext to implement the identification stage of search. In lieu of hypertext network navigation, one could use keyword search or full-text search. However, we have argued that there are persuasive reasons that explain why these techniques are unlikely to achieve the desired results. Hypertext, by contrast, offers significant advantages. *First,* developers can actively refine their search by inspecting software objects and by deciding the direction for further navigation. This ensures that developers are in control of the identification process, and, as a result, are likely to conduct a more thorough search. *Second,* the active nature of hypertext network navigation and object inspection will result in an increased awareness of repository contents. Hypertext-supported repository search, therefore, enables developers to become more familiar with the repository, which, in turn, reduces future search costs. *Third,* hypertext enables the exploration of a relatively large set of objects, and so the initial screening does not need to be very precise. The benefits of the navigational aids provided in typical hypertext tools are, therefore, likely to have a favorable impact on overall search costs, and, thus, improve the potential for reuse.

When identification concludes, the developer will have located and retrieved a small set of applicable objects that can be reused. More importantly perhaps, this process will yield information about whether there are objects with the appropriate functionality that are in the repository. This knowledge will assist the developer in coming to the right decision about whether to scratch build an object.

The ambiguity problems we referred to in Section 2 prompt additional discussion. Lexical ambiguity can be minimized by implementing a controlled vocabulary. When developers select search criteria they do so by picking from a set of given classifiers—a pull-down menu incorporating the feasible choices is a practical approach to communicate this constraint. Hypertext capabilities are also geared to enable developers to resolve the related problem of near matches. The various guided tours group together objects when they have similar functionality. Developers can easily inspect similar objects by following these guided tours.

6.3 Hypertext Illustration

Imagine a developer who is working with an ICE repository that contains the sample ICE RULES presented earlier. Further suppose that the developer needs a high level RULE to produce a report 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.

Screening concludes with the creation of hypertext guided tours linking the various rules. For each facet value that is shared by more than one RULE, 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. Fig. 6 shows portions of guided tours GT-1, GT-3, GT-6, and GT-10.

Next comes identification. Each of the guided tours created at the end of the screening step enables a developer to explore similar RULES, i.e., those that share the same classifier in a given facet. Because some of the guided tours intersect with each other, there are opportunities for a developer to move among exploration paths at intersection points. Fig. 7 depicts a portion of the derived hypertext network that shows the intersection of guided tours at RULE **CUST-ACCOUNTS** (RULE 7). Upon reaching that RULE, a developer 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 Fig. 6, RULE 7, **CUST-ACCOUNTS,** is related to RULE 16, **SUBORDI_CUST_LINK,** via GT-4, which groups *THREAD* 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 *THREAD* RULES guided tour.

A software developer can become disoriented if the resulting hypertext network is too complex. There are two potential sources of complexity: Either there are too many intersecting guided tours, or too many candidates for inspection (each candidate can spawn a guided tour). The first case only occurs if there are RULES that are classified under a large number of criteria. An inspection of Table 5 reveals that in our illustration the highest such number is five, which is manageable. Moreover, our interviews with developers indicated that rules classified under more than five categories could not exhibit sufficient modularity to be reusable[3]. The only remaining potential source of complexity is a large candidate pool. However, the size of the candidate pool can be controlled during screening by specifying precise search parameters. Thus, the user can control the complexity of the hypertext network. In addition, a reuse tool can monitor the results of screening and advise a software developer when search criteria are vaguely specified.

Our illustration demonstrates how to construct a hypertext network to support reuse from a classification of software objects. It is also readily automated: most of the information needed to classify ICE RULES is present in the repository. 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.

3. We also deduce from this the inadequacy of a classification schema where objects are classified under a large number of categories.

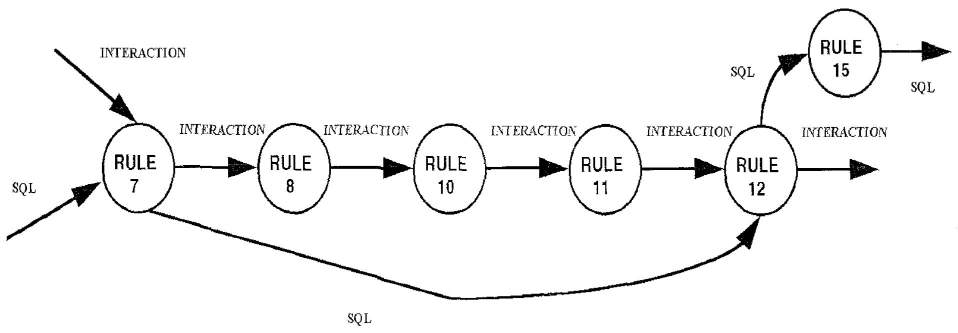


Fig. 5. Intersecting guided tours.

TABLE 6
RESULTS FROM SCREENING

	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 7
HYPERTEXT GUIDED TOURS GENERATED BY USING THE FACETED CLASSIFICATION

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

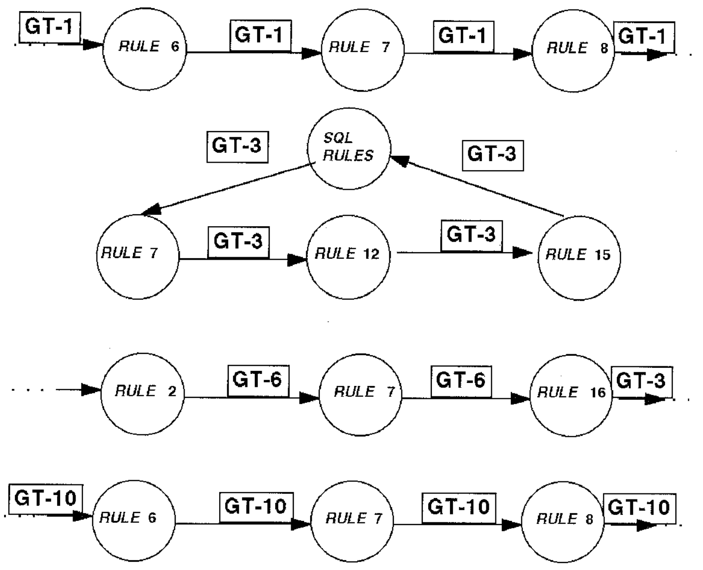


Fig. 6. Four guided tours containing RULE7.

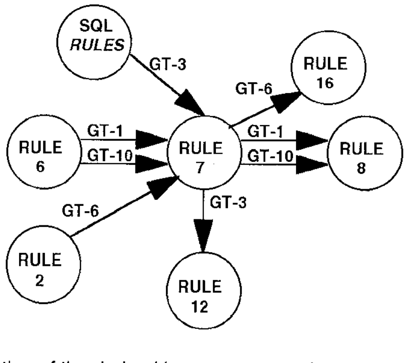


Fig. 7. A portion of the derived hypertext network.

7 The Prototype

A prototype reuse search support system to operate within ICE object repositories was built with the assistance of experienced software engineers. The prototype is the result of an ongoing collaborative corporate and university research effort to extend ICE's repository evaluation and software development tool set. The repository object search support facility is intended to run in client-server environments under OS/2, however, the prototype was built to run off a development repository stored locally on the client. It was implemented using ICE, and its objects were added to the local development repository.

7.1 ORCA and AMHYRST

Reflecting the conceptual basis of our model for reusable object search, the system consists of two primary tools, as shown in Fig. 8, that perform object classification and screening.

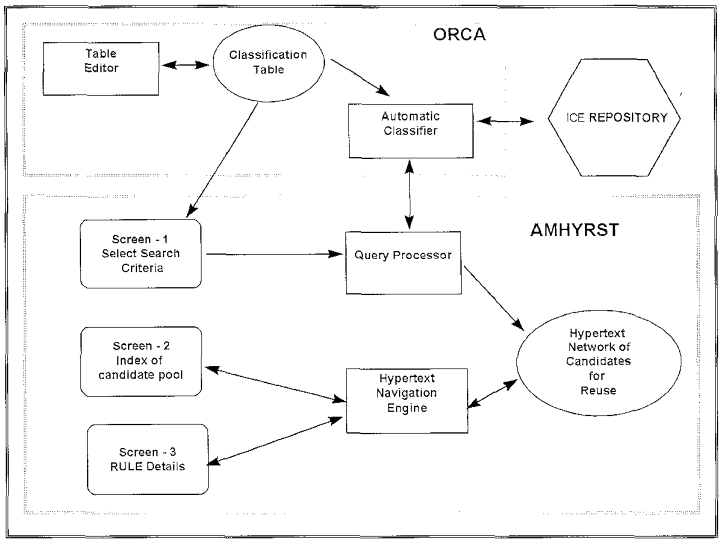
The first tool, called *ORCA* for *Object Reuse Classification Analyzer*, implements *screening* by combining an automated repository classifier and a query processor. This tool enables system developers to specify repository queries based on classification criteria that they supply. The classification is performed automatically by using information present within the ICE repository. Thus, developers are not required to manually classify the RULES. The second tool, *AMHYRST*, which stands for *AutoMated HYpertext Reuse Search Tool*, organizes the query results into a hypertext network and enables developers to inspect the query results via a hypertext engine.

7.2 Prototype Operation

We now depict a sample session with ORCA and AMHYRST, by presenting three screens that correspond to what the user sees when using the tools. The developer starts by selecting facet values from the three list-boxes shown in Fig. 9. Each list-box corresponds to one of the three classification criteria discussed earlier. The use of list-boxes ensures that developers choose valid classifiers, thereby eliminating word choice problems, and spelling or typing errors. In this example, the user has specified a search for RULES classified as *LEAF* and *SQL* in any business domain.

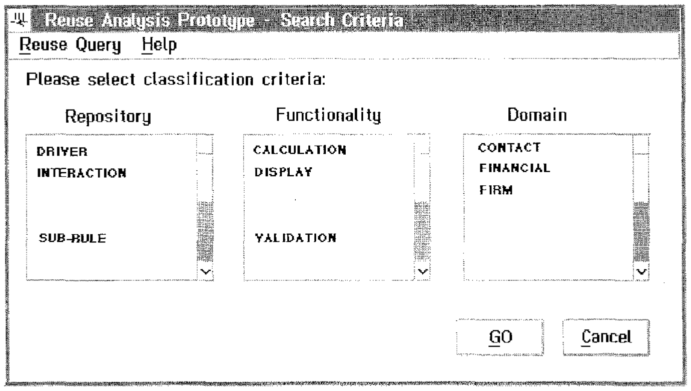
The names of all the RULES satisfying the criteria are retrieved into the screen shown in Fig. 10. The classification boxes shown enable the user to carefully define the objects to be explored. For example, by selecting *FIRM*, and clicking on the right (left) arrow, the developer is presented with the next (previous) *FIRM RULE* from the list. The guided tours to traverse are selected by clicking on a classification box, and the arrows determine the direction of navigation within the guided tour (recall that rules are order lexicographically within a guided tour). The developer can inspect a RULE more closely by double clicking on its name. This brings up the detailed screen as shown in Fig. 11.

The prototype system contains an algorithm that classifies objects based on information available in the repository. With each classifying facet for an ICE object, we associate a value that determines its membership in a given class.


ORCA
Table Editor
Classification Table
Automatic Classifier
ICE REPOSITORY
AMHYRST
Screen - 1 Select Search Criteria
Query Processor
Screen - 2 Index of candidate pool
Hypertext Network of Candidates for Reuse
Hypertext Navigation Engine
Screen - 3 RULE Details

Legend:
rectangle—processing function
oval—database function
hexagon—ICE repository
rectangle with rounded corners—input screen function

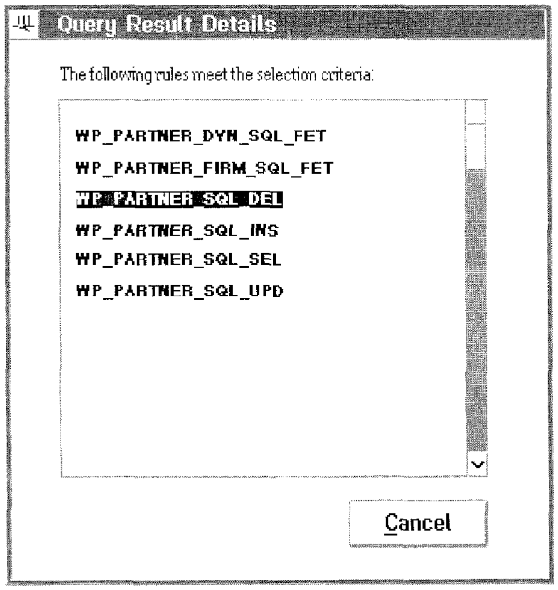
Fig. 8. An architecture for the prototype reuse search tool.



Please select classification criteria:

Repository	Functionality	Domain
DRIVER	CALCULATION	CONTACT
INTERACTION	DISPLAY	FINANCIAL
SUB-RULE	VALIDATION	FIRM

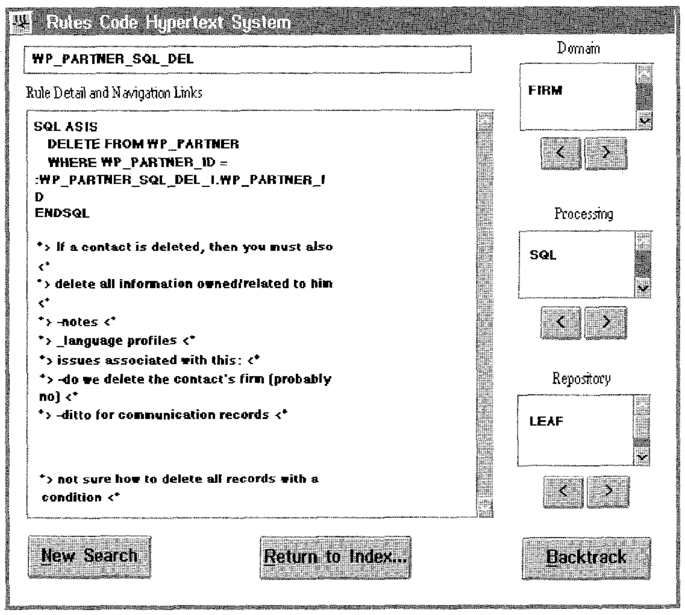
Fig. 9. Screen used to enter criteria for search.



The following rules meet the selection criteria:

WP_PARTNER_DYN_SQL_FET
WP_PARTNER_FIRM_SQL_FET
WP_PARTNER_SQL_DEL
WP_PARTNER_SQL_INS
WP_PARTNER_SQL_SEL
WP_PARTNER_SQL_UPD

Fig. 10. 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 <*
```

Fig. 11. Hypertext navigation to inspect contents of a RULE.

Table 8 shows a number of the classification methods that were used to implement the repository, business process and domain facets in the prototype. The criteria that are listed under "Method" are experimental; they are currently being refined by ICE developers. The automated classifier, ORCA, uses these methods to produce a set of inverted tables, one per classifier, containing all the RULES that fall under a classification. When a developer selects criteria for search using the classification screen, an SQL query is fired against these internal tables. Then, the results are retrieved into a temporary area that is used by AMHYRST to produce the hypertext network. Finally, a hypertext engine allows the developer to browse the network of guided tours to identify an appropriate rule for reuse.

7.3 Implementation Considerations and Limitations

Two important implementation concerns became evident through the development of the prototype. *First*, the criteria for object classification are likely to evolve over time, as the repository grows and the CASE tool set changes. As a result, ORCA will eventually need to accommodate changes in the classification schema. In the prototype, the criteria for classification were hard-coded, offering insufficient flexibility to address this problem. Adding new classifiers requires recompilation of significant portions of the prototype system. To overcome this limitation, we contemplate using a classification table with the format shown in Table 8. This would be used as an input parameter to ORCA, enabling it to be fine-tuned as useful new classifications are discovered. It will also be helpful to have facilities that enable this table to be edited. *Second*, we learned from the reactions of people who evaluated the prototype that fast response times are essential in true client-server ICE development. For example, slow response times for the classification of a server-based repository will lead developers to perceive excessive search costs. They then would be reluctant to use the search facility. A compromise that we are exploring is to have ORCA classify the repository off-line, on a periodic basis, so that the last classification can be accessed immediately. A second approach that we are exploring is to classify the repository incrementally. The benefits of fast access are perceived to outweigh the disadvantages of not including recently created RULE objects in the search process.

Currently, the prototype has limited hypertext navigational facilities, and is restricted to RULE objects and to a small set of classifiers. In spite of these restrictions, the prototype demonstrates the applicability of our reuse search model. It also demonstrates the feasibility of specifying an automated classification algorithm. Based on our experience, a reusable object search tool that does not incorporate this feature will be very inefficient to use. Another limitation of our prototype is that it places no structure on Stage 3, the decision phase. An important step towards improving a developer's ability to decide whether to build a new object is to provide an estimate of what it might cost to build an object with relatively similar functionality. For this reason, we are conducting additional research to determine standard costs for various kinds of repository object types, and how such costs could be estimated from existing repository information.

7.4 Contributions

The contributions of this research are threefold. We have proposed and illustrated an approach to automating classification of object repository objects in an integrated CASE environment. We have shown that hypertext technology, in conjunction with an repository-based application metamodel, provides a useful set of capabilities to expand developers' capabilities to search for reusable software objects. And, we have demonstrated how these observations can be used to design working reuse search support tools for real world software development.

TABLE 8
STORAGE OF CLASSIFICATION METHODS IN ICE TABLES

CLASSIFIER	METHODS	EXPLANATION
	REPOSITORY FACET	

DRIVER	SELECT FROM RULE_RULE
	WHERE INSTANCE_B = rule_id
		Look up TABLE RULE_RULE if the RULE uses another RULE

SUB-RULE	SELECT FROM RULE_RULE
	WHERE INSTANCE_A = rule_id
		Look up TABLE RULE_RULE if the RULE is used by another RULE

LEAF-RULE	SELECT FROM RULE_RULE
	WHERE INSTANCE_B = rule_id
	result is the EMPTY relation
		Look up in TABLE RULE _ RULE if the RULE does not use another RULE

	PROCESS FACET	

SQL	SELECT FROM RULE_FILE
	WHERE ENTITY_TYPE_B = "RULE"
	AND INSTANCE_B = rule_id
		Does this RULE use a FILE?

DISPLAY	SELECT FROM RULE_WINDOW
	WHERE ENTITY_TYPE_B = "RULE"
	AND INSTANCE_B = rule_id
		Look up RULE in the TABLE RULE_WINDOW

	DOMAIN FACET	

Domain X	SELECT FROM RULES
	WHERE DESCRIPTION= "%X%"
		Look up RULE name in RULES TABLE.

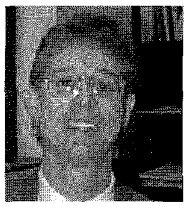
ACKNOWLEDGMENTS

The authors wish to acknowledge Marc Baric, Gene Bedell, Tom Eichner, Gig Graham, Tom Lewis, 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, and Howard Sloan for developing the prototype system. Finally, we wish to acknowledge Michael Bieber, Minder Chen, Hank Lucas, and Dani Zweig, whose comments influenced earlier drafts of this paper. All errors are the responsibility of the authors.

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, Dec. 1990.
[2] 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, Sept. 1991.
[3] 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," *Analytical Methods for Software Eng. Economics I*, T. Gulledge and W. Hultgren, eds. New York: Springer-Verlag, 1993.
[4] 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, Apr. 1993.
[5] M.J. Bates, "Subject Access in On-Line Catalogs: A Design Model," *J. Am. Soc. Information Sciences*, vol. 37, no. 6, pp. 357-376, Nov. 1986.
[6] B. Beckman, W. Van Snyder, S. Shen, J. Jupin, L. Van Warren, B. Boyd, and R. Tausworthe, "ESC: A Hypermedia Encyclopedia of Reusable Software Components," Jet Propulsion Laboratory, California Inst. of Technology, Pasadena, Sept. 1991.
[7] J. Bigelow and V. Riley, "Manipulating Source Code in Dynamic Design," *Hypertext '87 Proc.*, pp. 397-408, Chapel Hill, N.C., Nov. 1987.
[8] D.C. Blair and M.E. Maron, "An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System," *Comm. ACM*, vol. 28, no. 3, pp. 289-299, Mar. 1985.
[9] G. Booch, "What Is and Isn't Object-Oriented Design," *Ed Yourdon's Software J.*, vol. 2, no. 7-8, pp. 14-21, Summer 1989.
[10] M.L. Creech, D.F. Freeze, and M.L. Griss, "Using Hypertext in Selecting Reusable Software Components," *Hypertext '91 Proc.*, pp. 25-38, San Antonio, Tex., Dec. 1991.
[11] J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, vol. 20, no. 9, pp. 17-41, Sept. 1987.
[12] M. Chen, "A Model Driven Approach to Accessing Managerial Information: The Development of a Repository-Based Executive Information System," *J. Management Information Systems*, vol. 11, no. 4, pp. 33-63, Spring 1995.
[13] W.B. Frakes and T.P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," *IEEE Trans. Software Eng.*, vol. 20, no. 8, pp. 617-630, Aug. 1994.
[14] G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais, "The Vocabulary Problem in Human-System Communications," *Comm. ACM*, vol. 30, no. 11, pp. 964-971, Nov. 1987.
[15] 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.
[16] F. Garzotto, P. Paolini, and L. Mainetti, "Navigation in Hypermedia Applications: Modeling and Semantics," *J. Org. Comp.*, forthcoming.
[17] F.G. Halasz, "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems," *Comm. ACM*, vol. 31, no. 7, pp. 836-852, July 1988.
[18] S. Henninger, "Using Iterative Refinement to Find Reusable Software," *IEEE Trans. Software Eng.*, vol. 11, no. 5, pp. 48-59, Sept. 1994.
[19] IEF Technical Description: Methodology and Technology Overview, *TI Part #2739900-8120*, Texas Instruments, Plano, Tex., Aug. 1992.
[20] T. Isakowitz, "Hypermedia in Organizations and Information Systems: A Research Agenda," *Proc. 26th Hawaii Int'l Conf. Systems Science*, vol. 3, pp. 361-369, Maui, Hawaii, Jan. 1993.
[21] J. Karimi, "An Asset-Based Systems Development Approach to Software Reusability," *MIS Quarterly*, vol. 14, no. 2, pp. 179-198, June 1990.
[22] P. Kerola and H. Oinas-Kukkonen, "Hypertext System as an Intermediary Agent in CASE Environments," *The Impact of Computer Supported Technologies on Information Systems Development*, K.E. Kendall, K. Lyytinen, and J. DeGross, eds., pp. 289-313. New York: North-Holland, 1992.
[23] Y. Kim and E. Stohr, "Software Reuse: Issues and Research Directions," *Proc. 25th Hawaii Int'l Conf. Systems Science*, vol. 4, pp. 612-623, Maui, Hawaii, Jan. 1992.

[24] J. Nielsen, *Hypertext and Hypermedia*. New York: Academic Press, 1990.
[25] J. Nielsen, "Navigating Through Hypertext," *Comm. ACM*, vol. 33, no. 3, pp. 297-310, Mar. 1990.
[26] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse," *Comm. ACM*, vol. 34, no. 5, pp. 89-97, May 1991.
[27] R.H. Trigg, "A Network-Based Approach to Text Handling for the On-Line Scientific Community," Computer Science Technical Report no. TR-1346, Dept. of Computer Science, Univ. of Maryland, College Park, 1983.
[28] W. Van Snyder, "Software Classification and Retrieval," Technical support package for NASA Technical Brief NPO-18530, NASA Techical Briefs 17, 8, Item 27, Aug. 1993.
[29] R.J. Welke, "The CASE Repository: More Than Another Database Application," *Challenge and Strategies for Research in Systems Development*, B. Cotterman and J.A. Senn, eds., pp. 181-218. New York: John Wiley, 1992.
[30] 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.
[31] W.A. Woods, "What's in a Link: Foundations for Semantic Networks," *Representation and Understanding: Studies in Cognitive Science*, D.G. Bobrow and A. Collins , eds., p. 82. New York: Academic Press, 1975.



Tomás Isakowitz received his BSc in mathematics at the Hebrew University of Jerusalem, his MSc in mathematics at the University of California at Santa Barbara, and his MEng and PhD in computer science at the University of Pennsylvania. He is an assistant professor of information systems at New York University Stern School of Business. His research interests are hypermedia technology and its applications, decision support and temporal databases. Professor Isakowitz taught at New York University's Stern School of Business, the International University of Japan, and the University of Pennsylvania, and was a visiting scholar at Stanford Research Institute. Dr. Isakowitz' software engineering research interests focus on exploring new paradigms and methodologies to facilitate software development, encourage software reuse, and encourage solid system construction. He is actively involved in hypertext research, and has written extensively about the design and development of hypertext/hypermedia applications. His publications have appeared in the *Journal of Management Information Systems, Communications of the ACM, Decision Support Systems, ACM Transactions on Database Systems, ACM Transactions on Office Information Systems, Decision Support Systems*, and elsewhere. He currently serves on the editorial boards of the *Journal of Management Information Systems*, the *Journal of Electronic Commerce*, and as a special issue editor for the *Journal of Organizational Computing* and the *Communications of the ACM*. Dr. Isakowitz has worked as a consultant for several international, and is actively involved in the academic and practical issues involving the design of hypermedia applications.

Robert J. Kauffman completed a PhD in industrial administration at Carnegie Mellon University, and also holds degrees from Cornell University and the University of Colorado, Boulder. He is an associate professor of information systems and decision sciences at the Carlson School of Management, University of Minnesota. He taught at New York University's Stern School of Business and the University of Rochester's Simon Graduate School of Management, and was a visiting scholar at the Federal Reserve Bank of Philadelphia. Dr. Kauffman's software engineering research interests focus on exploring new methods and perspectives for estimating software costs, evaluating application performance, and understanding the leverage that new software development techniques provide in maximizing the value of the firm. His broader research agenda as a business technologist involves assessment of the value of information technology investments and applications of information technology in the financial services arena, using techniques from economics, finance, and management science. His publications have appeared in *Decision Science, EFT Today, IEEE Transactions on Software Engineering, Information and Management, Information and Software Technologies, Journal of Management Information Systems, Journal of Strategic Information Systems, MIS Quarterly*, and elsewhere. He currently serves on the editorial boards of *Information Systems Research*, the *Journal of Management Information Systems* and the *Journal of Electronic Commerce*, and as a special issue editor for the *Journal of Organizational Computing* and the *Communications of the ACM*.