

**Facilitating Software Reuse by Structuring the SPS User
Interface Management System's Software Library According
to Programmer Mental Models**

by


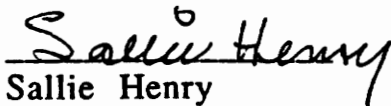
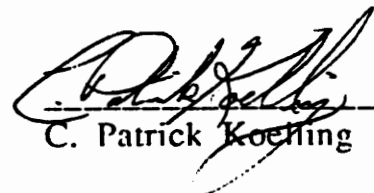
Joseph A. Jenkins

Dissertation submitted to the Faculty of the Virginia Polytechnic
Institute and State University in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
Industrial and Systems Engineering

APPROVED:



Harry L. Snyder, Chairperson


Robert Beaton
Dennis Kafura
Sallie Henry
C. Patrick Koehling

May, 1994
Blacksburg, Virginia

C.2

LV
5655
V956
1994
J465
L.2

Facilitating Software Reuse by Structuring the SPS User Interface Management System's Software Library According to Programmer Mental Models

by
Joseph A. Jenkins
Committee Chairperson: Harry L. Snyder
Industrial and Systems Engineering
(ABSTRACT)

This study evaluates three different ways of structuring a software library for an object-oriented system. The traditional class/subclass tree (C/S) is used as well as two methods from the mental model literature: hierarchical cluster analysis (HCA) of sorting data and modal block clustering (MBC) of attribute rating data (Shurtleff, Jenkins, and Sams, 1988; Tullis, 1985).

Also examined in this context are two software metrics: depth-in-inheritance-tree (DIT) and response-for-class (RFC) (Lei, 1991, 1993). These two metrics had been found by Lei to correlate with the ease of maintenance of software. It was conjectured that they might also be useful in the study of mental model methods for software.

Finally, student and professional programmers are explicitly compared. There has been much debate on the applicability of software-related data generated from student subjects but little research on the topic.

The results indicate that subject performance with the MBC representation was worse than with the C/S representation. Also found was that performance with the HCA representation was not

sufficiently better to justify the effort involved in creating the new representation.

Student programmers were found in this study to be no worse than professional programmers and thus appear to be acceptable substitutes for professional programmers in a class-based search task. This study's results indicate that student response time to locate a class could be used as the lower boundary for professional programmer class locating time. The student error rates, however, could be used as the upper boundary for professional programmer error rate performance in class locating tasks.

A "middle" problem was also found. Classes in the middle of a library representation proved to be more difficult to locate as measured by time and error rate.

Finally, a higher RFC was found to increase search time for a class when it was in the middle of a library representation. A higher RFC also reduced programmer reusability ratings of a class and the programmer's confidence in the reusability rating.

ACKNOWLEDGEMENTS

I would like to extend my appreciation to BNR, Inc. for providing the people and resources to enable this study to occur.

I would like to thank my committee members for indulging me. They allowed me the freedom to pursue the ideas that I am interested in and this is no small privilege.

I would also like to thank my advisor, Dr. Harry Snyder, for his patience. It is my hope that I can contribute something of worth that justifies his good humor and understanding.

A big thanks goes to Reni Jenkins. She deserves most of the credit for my finishing by combining a motivating mixture of reassurance, encouragement, and threats. Clearly, I would never have finished without her.

Finally, my most special thanks go to Joseph and Margaret Nagy for instilling in me the desire for something more.

TABLE OF CONTENTS

INTRODUCTION	1
Current Research.....	3
Mental Models.....	5
Methods of Generating Models.....	14
Class/Subclass Tree.....	15
Multidimensional scaling of pairwise similarity ratings.....	16
Hierarchical cluster analysis of sorted data.....	18
Modal block clustering of attribute rating data.....	20
Programmers: Student Versus Professional.....	24
Summary of Introduction.....	27
Research Objectives.....	28
METHOD	29
Experimental Design	29
Subject.....	32
Materials and Apparatus.....	34
Procedure of Task 1.....	36
Time requirement.....	36
Task sequence.....	36
Dependent variables.....	39
Procedure of Task 2.....	39
Time requirement.....	39
Task sequence.....	39
Dependent variables.....	40
Experimental Conclusion.....	41
RESULTS.....	42
Task 1.....	42
Task 1 RT.....	42
Task 1 Errors.....	54
Task 2.....	64
Task 2 RT.....	64
Task 2 Ratings.....	68
Task 2 Rating RT.....	70
Task 2 Errors.....	76
DISCUSSION.....	84
Task 1	84
Task 1 RT.....	84
Task 1 Errors.....	90
Task 2.....	93
Task 2 RT.....	93

Task 2 Rating.....	95
Task 2 Rating RT.....	96
Task 2 Errors.....	98
CONCLUSIONS	102
REFERENCES	105
APPENDIX A.....	109
APPENDIX B	112
APPENDIX C.....	117
APPENDIX D	121
VITA	123

LIST OF TABLES

Table 1. Eight Generic Mental Model Types	1 0
Table 2. ANOVA Summary Table for Task 1 Reaction Time	4 3
Table 3. Mean Response Times for the Representation Conditions, Task 1	4 4
Table 4. Mean Response Times for the DIT Conditions, Task 1...	4 6
Table 5. Mean Response Times for the RFC Conditions, Task 1...	4 7
Table 6. Mean Response Times for the DIT X Representation Interaction Conditions, Task 1	4 8
Table 7. Newman-Keuls Post-Hoc Analysis for the DIT X Representation Interaction for Response Times, Task 1	4 9
Table 8. Mean Response Times for the RFC X Programmer Interaction Conditions, Task 1	5 1
Table 9. Mean Response Times for the DIT X RFC Interaction Conditions, Task 1	5 2
Table 10. Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Response Times, Task 1	5 4
Table 11. ANOVA Summary Table for Task 1 Error Rate	5 5
Table 12. Mean Error Rates for the Representation Conditions, Task 1	5 6
Table 13. Mean Error Rates for the DIT Conditions, Task 1	5 7
Table 14. Mean Error Rates for the RFC Conditions, Task 1	5 9
Table 15. Mean Error Rates for the RFC X Representation Interaction Conditions, Task 1	5 9
Table 16. Newman-Keuls Post-Hoc Analysis for the RFC X Representation Interaction for Error Rates, Task 1	6 1
Table 17. Mean Error Rates for the DIT X RFC Interaction Conditions, Task 1	6 2
Table 18. Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Error Rates, Task 1	6 3
Table 19. ANOVA Summary Table for Task 2 Reaction Time	6 5
Table 20. Mean Response Times for the Programmer Conditions, Task 2	6 6

Table 21. Mean Response Times for the DIT X RFC Interaction Conditions, Task 2	6 6
Table 22. Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Response Times, Task 2.....	6 8
Table 23. ANOVA Summary Table for the Task 2 Ratings	6 9
Table 24. Mean Rating Times for the RFC Conditions, Task 2.....	7 0
Table 25. ANOVA Summary Table for the Task 2 Rating Times...	7 1
Table 26. Mean Rating Times for the DIT Conditions, Task 2.....	7 2
Table 27. Mean Rating Times for the RFC Conditions, Task 2.....	7 2
Table 28. Mean Rating Times for the DIT X RFC Interaction Conditions, Task 2	7 4
Table 29. ANOVA Summary Table for Task 2 Errors.....	7 7
Table 30. Mean Error Rates for the RFC Conditions, Task 2.....	7 8
Table 31. Mean Error Rates for the DIT X RFC Interaction Conditions, Task 2	7 8
Table 32. Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Error Rates, Task 2.....	8 0
Table 33. Mean Error Rates for the DIT X RFC X Programmer Interaction Conditions, Task 2.....	8 1
Table 34. Newman-Keuls Post-Hoc Analysis for the DIT X RFC X Programmer Interaction for Error Rates, Task 2.....	8 3

LIST OF FIGURES

Figure 1. Adapted from Norman (1983, 1986) to show the different representations of a system.....	6
Figure 2. Three views of concepts.....	12
Figure 3. Top level between-subjects portion of the mixed design.....	30
Figure 4. Diagram of the DIT X RFC part of the design seen by all subjects.....	31
Figure 5. A representation of the computer screen for the tasks.....	38
Figure 6. Mean response times of the representations with standard error bars.....	44
Figure 7. Mean response times of the DIT levels with standard error bars.....	46
Figure 8. Mean response times of the DIT X Representation interaction with standard error bars.....	48
Figure 9. Mean response times of the RFC X Programmer interaction with standard error bars.....	51
Figure 10. Mean response times for the DIT X RFC interaction with standard error bars.....	53
Figure 11. Mean errors for the Representations with standard error bars.....	56
Figure 12. Mean errors for the DIT with standard error bars.....	58
Figure 13. Mean errors for the RFC X Representation interaction with standard error bars.....	60
Figure 14. Mean errors for the DIT X RFC interaction with standard error bars (Task 1).....	62
Figure 15. Mean response times of the DIT X RFC interaction with standard error bars (Task 2 RT).....	67
Figure 16. Mean rating times for the DIT with standard error bars.....	73
Figure 17. Mean rating times for the DIT X RFC interaction with standard error bars (Task 2 Rating RT).....	75

Figure 18. Mean errors for the DIT X RFC interaction
with standard error bars (Task 2 Errors).....79

Figure 19. Mean errors for the DIT X RFC X Programmer
interaction with standard error bars81

INTRODUCTION

Software reuse is the process of using previously existing software units to build new working programs. As a field of study, software reuse is generating a great deal of attention and work because of rising needs for increased productivity and increased quality. Software reuse activity is encouraged in many systems through the use of procedures, modules, and objects. There are many reasons to encourage software reuse, but among the most important are reduced development costs, increased reliability, and shorter development cycles (Davis, Bersoff, and Comer, 1988; McClure, 1992)

Unfortunately, a problem encountered by the user (typically a programmer) in many software systems that are based on the principle of software reuse is locating the relevant piece of code for a project. Good examples of current software reuse-oriented environments are Smalltalk, C++, and Lisp-based systems.

The Texas Instruments (TI) Explorer Lisp platform has approximately 12,000 functions and 15,000 flavors (flavors are similar to classes). Unfortunately, the large software library on the TI system created code location problems that were never solved. Thus, the information retrieval problem undermined the intended purpose of creating a reusable software library.

The size of the TI system and the resulting search problems are also beginning to appear in newer object-oriented systems. In particular, the problems are appearing in systems used by large development groups that are creating a large stockpile of classes (G. A. Moore, personal communication, January, 1992).

A number of researchers have noted that the inability to find existing components is a difficulty that needs to be addressed for successful software reuse (Davis et al., 1988; Grabow and Noble, 1988; Prieto-Diaz, 1987; Prieto-Diaz and Freeman, 1987). Prieto-Diaz and Freeman (1987) noted that the key to dealing with code accessibility (i.e., locating or finding) is a good classification scheme. Thus, a good method to organize a software library is an important part of creating a system that facilitates software reuse.

Domain analysis (Prieto-Diaz, 1987) is one recent attempt in the software engineering community to analyze an environment, problem, and/or information space so that the software components in a system can be logically arranged to ease the information search-and-retrieval problem (among other things). The success of domain analysis is mixed at best because it is essentially an exhaustive, sometimes ad hoc (Tracz, 1992), examination of some domain (i.e., area of application).

Techniques from knowledge acquisition have been used to help extract the logical arrangement of information from experts. Often the techniques are heavily dependent on the "craft-like" abilities of the practitioner.

If the domain analysis practitioner is "good" at what he does, then the results are useful. On the other hand, if the domain analysis practitioner is not so "good," then the results can be worthless. Clearly, a more repeatable and stable methodology is needed to become more engineering-oriented and to move away from the craft aspect.

Current Research

The motivation for the current research is the idea that much of the human information search-and-retrieval problem is due to an inappropriate organization of the software library. The typical software library is organized according to machine or software designer needs and not the software end-user needs. This research examines and evaluates three information organization methods that are based on different data gathering and clustering techniques.

The three organization methods are respectively based on three different types of mental models. Mental models, briefly, are representations of how information is structured and stored in human memory. In particular, this research is concerned with

programmer mental models of software libraries. A good information organization method will effectively represent the programmer's mental model. The hypothesis is that information organization based on the programmer's mental model will result in an increase in the efficiency (as measured by time and error rates) by which programmers can locate relevant code.

The current research deals with classes in an object-oriented system. Thus, the classes in an object-oriented system are treated as patterns to be classified for a human information processing/search task.

Traditionally, classes are usually put into tree representations generated from class/subclass relationships; thus, the current research also considers class/subclass trees. Other methods may be more advantageous for searching, however, so two human mental model methods are also employed to organize the library of objects. A more complete discussion of mental models is contained in a later section.

Another interesting parameter when dealing with mental models is level of subject experience/expertise. For programming, one often mentioned but rarely studied aspect is student versus professional programmers. Much software research is conducted with student subjects. Many people, however, question whether

student programmers are truly comparable to professional programmers. The current research incorporates student and professional programmers as a variable to be studied.

Another tool used currently in software engineering is measuring aspects of programs by applying software metrics. Various metrics have been studied and found to correlate to programmer errors and ratings of programming difficulty (e.g., Lei, 1991; Lei, 1993). Software metrics, therefore, are included in this research to capture and control program parameters of interest. Specifically, the Depth-In-Inheritance-Tree (i.e., DIT) and the Response-For-Class (i.e., RFC) metrics are used. The DIT metric is simply the location of a class within a class/subclass tree (Chidamber and Kemerer, 1991). The RFC metric is a count of all the local methods in a class and all the methods that the local methods call (Chidamber and Kemerer, 1991). Both of these metrics are elaborated on in the Methods section.

Mental Models

Norman (1983, 1986), in discussing mental models, stated that there are four different representations of a system that need to be recognized.

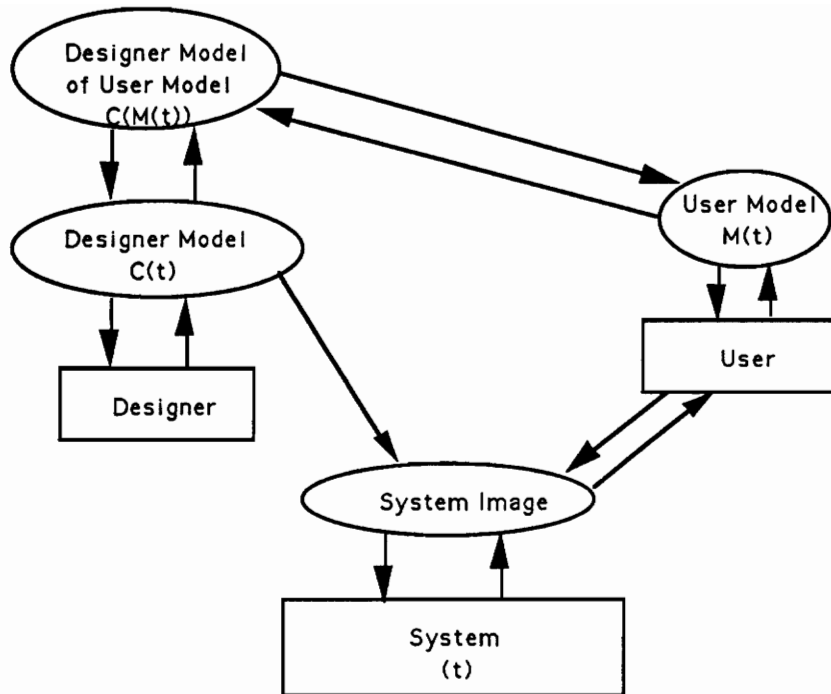


Figure 1. Adapted from Norman (1983, 1986) to show the different representations of a system.

The target system (t) is the first representation. This is the actual (i.e., physical) system.

The second representation is the conceptual or designer's model (C(t)). The conceptual model is created by the designer to provide a "good" representation of the target system. The critical features are accuracy, consistency, and completeness. The conceptual model should also be easily learned, contain the system's functionality, and be easily used.

The third representation is the user's mental model of the target system (M(t)). The user's mental model is mostly concerned with function and not with accuracy, consistency, or completeness. The user's mental model is built up through repeated interaction with the target system, usually within the context of performing some task. Limitations are imposed on the user's mental model through lack of background knowledge, experiences (e.g., Norman, 1983), and human information processing capabilities (e.g., Wickens, 1984).

The fourth and final representation is the designer's conceptualization of the user's mental model (C(M(t))). This "model of a model" is used by the designer to anticipate user problems.

Norman (1983) also described a system image. The system image is based on the designer's conceptual model. The system image includes all aspects of how the system interfaces with the user (e.g., user interface, documentation, technical help) and ultimately affects the mental model of the user.

The five components cohesively describe the layout of a system from the perspective of mental models (i.e., what is going on in the minds [and reality] of the various participants in system design and usage). The literature on mental models is quite large and varied, but the definition of a mental model is fragmented and vague at best

(Card, Moran, and Newell, 1983; Gentner and Stevens, 1983; Helander, 1988; Norman, 1986). Norman specified which participant's mental model is being discussed and referred to the user's mental model as THE mental model. All the other participant's models have different labels. This research uses Norman's (1983) distinctions and labels to facilitate comprehension.

Norman also made six observations on mental models that may prove important in interpreting the results of the proposed study. First, he noted that mental models are incomplete. This conclusion stems from the unlikelihood of a user remembering all of a system or even using all of a system.

The second observation is that users have great difficulty in "running" their model. "Running" is used in the sense that using a model is like "running a program" in the user's mind. The mental model can be "executed" to find out what will most likely happen (i.e., to predict) as well as to generate a course of action. The difficulty in "running" the model may also stem from the model not being an organized "program" but just a loose collection of beliefs, feelings, and superstitions (i.e., the model is not a mechanistic system). In other words, there is nothing to "run."

Third, users forget even the parts of the system that they have used. Forgetting creates an unstable mental model. New details are

learned and added, and old details that have not been used recently are forgotten (i.e., "deleted").

Fourth, mental models are not strictly bounded. Similar items and details are confused, interchanged, and merged.

Fifth, mental models contain many superstitions, beliefs, feelings, and guesses. Many of these "shortcuts" are retained because they trade (and reduce) mental effort for some slight increase in physical effort.

Sixth, mental models are parsimonious. Users will often apply general simplified rules even if the rules result in more physical actions because the mental complexity of a task is reduced.

These six observations are relevant to the current study because they discuss incompleteness, changing models, and boundaries/confusability. All these observations will make an impact on how a software library should be organized to deal with these features of mental models.

Young (1983), in another study of mental models, generated a list of eight generic possible types of mental models. Although the list is in no way suggested to be complete, it does serve as a good starting point. The eight types of mental models are listed in Table 1.

TABLE 1

Eight Generic Mental Model Types (Young, 1983)

1. Strong Analogy: this is a metaphor - X is like Y
 2. Surrogate: mechanistic model of how something works
 3. Task/Action Mapping: mediates between task and actions to be performed by user
 4. Coherence: memory schema - facts are organized around some schema
 5. Vocabulary: based on a symbolic coding - familiar things are easier to learn (assimilated) than are new things (creation)
 6. Problem Space: describes problem space in which problems are solved
 7. Psychological Grammar: based on psycholinguistics - production rules and structures
 8. Commonality: model of processes and underlying data structure used
-

The type of model of most interest for this research is the coherence model. The coherence model allows one to connect mental models to human categorization theory in an empirically supported manner (to be covered below). Ultimately, one can then organize a software library according to attributes and relations defined by an

empirically derived mental model that is consistent with research from the field of human categorization abilities.

Thus, the mental model methods used to reorganize the software library will be repeatable and empirical methods. These methods are also compatible with human categorization data from the fields of psychology, cognitive science, and human factors engineering.

The coherence model is also interesting because it can be described mathematically. Various properties can be manipulated and examined in a more structured manner. Finally, the coherence model, depending on the data collection method used, can be automated. The last point is critical if one wants to create an organizational method that can dynamically adapt to changing software libraries.

The coherence model, as noted in Table 1, is concerned with schemas and organizing items around schemas. Thus, items are arranged and related in a logical manner according to a schema. The relatedness used for the organization is similarity. Similarity is a key concept because it relates the underlying structure and organizing method of mental models to human categorization abilities.

The models of human categorization fall (Figure 2) into three distinct camps: the classical view, the prototype/probabilistic view, and the exemplar view (Smith and Medin, 1981).

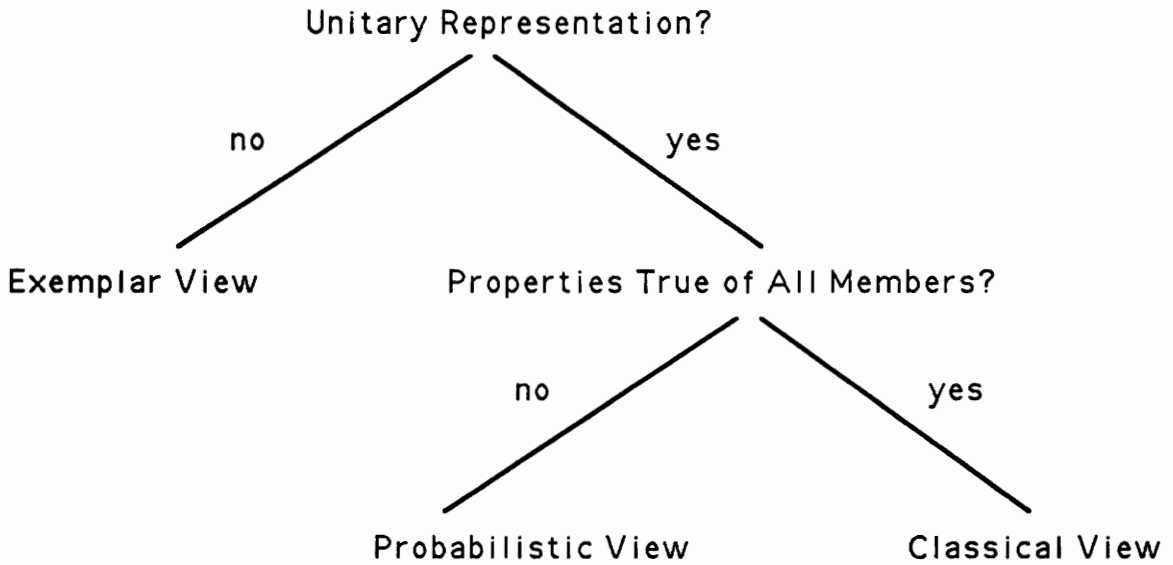


Figure 2. Three views of concepts (from Smith and Medin, 1981, p. 6).

The classical view of human categorization is that a set of common properties is shared by all the instances of a concept and that these common properties are necessary and sufficient to define a concept. More recent theories are the prototype and exemplar theories.

Prototype theory (Rosch and Mervis, 1975) holds that all the instances of a concept differ in the amount or degree to which they share properties. Thus, each instance of a concept differs in respect to how well it represents a concept. From this one can see that the representation of a concept is a sort of central tendency of the instances' properties and not a set of necessary and sufficient defining properties and degree of properties.

The exemplar view of human categorization states that there is not even a single best representation (i.e., best case or prototype) of a concept. Concepts can only be represented by specific examples and no "average" instance or prototype is formed (Smith and Medin, 1981).

The two more recent theories, prototype and exemplar, while slightly different, use the same example-based and feature-based approach. Each also supports the notion of concept instability (the classical view does not) which, as Norman (1983) noted, is an important feature of mental models. Thus, this research uses the prototype-based and exemplar-based theories as a model of human categorization. Which one does not matter for the purposes of this research because the goal of both theories is to show the relations between concepts using a group of examples, attributes, or features.

An important distinction is made in these two theories between component and holistic features. Component properties are a piece or part of a concept. They help describe a concept but are not a concept (e.g., parts of a bicycle). Holistic properties give a complete description of a concept. A template of the ideal bicycle would be an example of a holistic property. Because of the vagueness of holistic properties (e.g., what is an ideal example and how does one define it?), only component properties are used, where appropriate, to describe the objects for this study.

The theoretical underpinnings supplied by prototype and exemplar theories allow the study to proceed with certain assumptions (e.g., explicit attributes) of how best to represent the objects as well as how to interpret the findings. Prototype and exemplar theories also justify the use of clustering algorithms to organize similarity and/or attribute data. The reason is because the clustering algorithms may imitate the modeled human system (in theory) or at least the clustering algorithm results may be similar to human mental representations of concepts.

Methods of Generating Models

Each of the clustering methods (as well as the class/subclass tree) makes certain assumptions about the data being analyzed. The different methods also organize and display the resulting data

differently. These obvious statements, however, play a large part in selecting possible useful methods for organizing a software library. The following sections review each of the three methods to be used to organize software libraries. These methods are the class/subclass tree, hierarchical cluster analysis of sorted data, and the modal block clustering of attribute rating data. A fourth method, multidimensional scaling, is also discussed but rejected due to methodological practicalities.

Class/subclass tree. The typical manner in which classes and subclasses are represented in an object-oriented system (especially for systems that do not allow multiple inheritance) is via a class/subclass tree. The tree is based on the concept that a class can inherit data and behavior from another class. Typically, the inherited data are in the form of instance variables and the inherited behavior is in the form of methods. Thus, the class inheriting the data and behavior is a subclass (also called a child class) of the class (also called the parent class or super class) from which it inherits the data and behavior. A class with no superclass (i.e., parent) is a root or base class.

The creation of new subclasses occurs when the existing classes do not successfully address a programming need. The programmer finds the class that most closely solves the problem and then creates a new subclass (based on that class). The new subclass is then

adjusted to solve the problem by making any necessary changes in the methods and instance variables from the parent class. In other words, the programmer is programming by difference (i.e., how is the current requirement different from what I already have?).

The classes and subclasses are, therefore, organized in a tree form starting with the one underlying root class (if applicable, sometimes there are multiple root classes) and branching out into the numerous subclasses. The tree is, technically, organized by the subclasses' reuse of the methods and instance variables of the superclasses.

Multidimensional scaling of pairwise similarity ratings.

Multidimensional scaling has been applied a number of times with at least some degree of success for representing various types of mental models (Caramazza, Hersh, and Torgerson, 1976; Cooke, Durso, and Schvaneveldt, 1985; McDonald, Stone, Liebelt, and Karat, 1982). Of much interest is the claim by McDonald, Dearholt, Paap, and Schvaneveldt (1986) that multidimensional scaling is particularly useful for interfaces that would benefit from a spatial organization of some group of items. Because a graphical representation of a software library is under consideration in this dissertation, the multidimensional scaling approach appears promising.

Unfortunately, however, multidimensional scaling is unusable as a practical method in system design. Multidimensional scaling is unusable because it involves an exhaustive comparison of all possible pairs of items. The data collection effort thus increases at a rate of $(1/2[n(n-1)])$. For example, a group of 100 (n) classes to be compared would result in 4950 comparisons. Assuming that a person could make six comparisons per minute (this is optimistic) and works continuously, then he would be done in 13 hours and 45 minutes. If there were 500 classes, then there would be 124,750 comparisons.

Clearly, this is not a viable or practical approach. Thus, more efficient and economical alternatives are examined.

The method to create a multidimensional scaling representation is included below for completeness and to show the logical continuity between this technique and its more efficient relatives.

A multidimensional scaling representation of a class library is created as follows (Cooke, Durso, and Schvaneveldt, 1985; Curtis, 1989):

1. A list of all classes and subclasses is created.
2. Subjects rate the similarity of all possible pairs of classes and subclasses on scale from 1 to 9.

3. The similarity ratings are then averaged.
4. The similarity ratings are submitted to the multidimensional scaling analysis program.

A network or tree representation can then be made by lumping all clustered items into a node. Any particular node can then be linked to its nearest neighbor(s) using some distance threshold to limit the number of links.

Hierarchical cluster analysis of sorted data. The basic method for this approach was described by Tullis (1985). Tullis applied a hierarchical cluster analysis to data from a sorting task in his study of menu structure creation for a large system. The same method is applied in this research for the organization of classes and subclasses.

The goal of Tullis (1985) was to create a menu structure that accurately reflected the user's idea of how the system's functions were logically related. The current study is concerned with finding a method that accurately reflects the user's idea of how classes and subclasses are logically related. Clearly, the method proposed by Tullis (1985) can be used to create a user's view of a software library.

Tullis (1985) noted that the usage of traditional techniques (i.e., making the subject judge the similarity of all possible pairs of items)

was unrealistic for large systems due to the large number of items (and hence time and money) involved. Thus, he proposed reducing the data by first having the subject sort the items into stacks of cards. All items in one stack were then relabeled and represented as single items. The reduced data were then sorted in stacks again by another set of subjects. A distance matrix was then created from the sorted items and the distance matrix was then submitted to a hierarchical cluster analysis.

The six steps in the complete procedure, as adapted for the current study, are listed below.

1. One-sentence descriptions of each of the classes and subclasses are written on separate index cards.
2. Pilot subjects who are familiar with the software library (i.e., the classes and subclasses) sort the cards into groups based on similarity. The new groups are reduced by retaining only one function to represent the entire group.
3. The reduced set of cards is then sorted by another set of subjects. The subjects are told to sort the cards into logically related groups and then to assign a label to the group they have created. "Logically related" is not defined for the subjects.

Next, the same subjects sort the new groups with their labels into yet larger groups. Tullis (1985) called these "categories." The sort criterion is, again, logical relatedness. Once again, "logically related" is not defined for the subjects.

4. In this step, a perceived distance matrix is created. The distance is for all pairs of cards. Decision rules are as follows:
 - a. Any two cards placed in the same group are assigned a distance value of 0.
 - b. Any two cards placed in different groups but the same category are assigned a distance value of 1.
 - c. Any two cards placed in different categories are assigned a distance value of 2.
5. The distance matrices are summed across all subjects.
6. The summed distance matrix is submitted to a hierarchical cluster analysis (i.e., the minimum method).

The tree resulting from the analysis is then graphically represented and used as a library structure to be searched in the experiment.

Modal block clustering of attribute rating data. Shurtleff, Jenkins, and Sams (1988) proposed an alternative methodology to

hierarchical cluster analysis for the construction of menu systems. It is believed that their modal block clustering may also be appropriate for software library organization.

Modal block clustering defines clusters based on the pattern of common attributes. It is important to note that the attributes used for the clustering are explicit. There are no matrices of "similarity" or "dissimilarity" judgments. The clustering is based on explicit attributes that are rated as "present" or "not present."

There are five main advantages in using modal block clustering. The first advantage is iterative cluster definition. Suboptimal splits are not propagated through the tree as they are with hierarchical cluster analysis. The modal block cluster analysis algorithm iterates until the optimal split is obtained.

A second advantage is that overlapping clusters result from modal block clustering. The overlap of clusters allows for the objective definition of redundancy. Empirical support for the superiority of redundant menus over nonredundant menus was given by Roske-Hofstrand and Paap (1986). It is postulated that overlap will also prove advantageous in the class search and retrieval process.

A third advantage is that a modal block cluster analysis requires less work in data collection (i.e., number of classes X

attributes versus $(1/2[n(n-1)])$ comparisons for pairwise similarity ratings based approaches). This feature is important because a highly involved method that requires a lot of time, effort, and money will not be used. Schedules and budgets are often deemed more important than creating quality products. Thus, method efficiency is a key point to encourage system designer usage of a method.

A fourth advantage is that fewer clusters are created than with hierarchical clustering approaches. The numerous clusters that result from a typical hierarchical cluster analysis are why Tullis (1985) employed three levels of sorting to reduce the number of possible clusters.

The fifth, and possibly most important, advantage of a modal block clustering approach is that an attribute profile is used and not a similarity rating. Attributes can be linked to concrete features of classes (the number and type of methods, specific toolbox calls, software metric values, etc.) while similarity ratings are abstract at best. The similarity ratings of individual raters will be influenced by their specific, limited usage patterns of a system. It is also questionable to have a rater rate a class that he has never used (i.e., what is the basis for the rating?). An attribute, however, is specific, limited, and can be specified a priori by the system designer. Also, an attribute rater does not have to be expert in the use of a class to rate it on an attribute. This is because the question is not "how

similar is that class is to another class?" but only "does that class have some feature?" which can be looked up in the code or manuals.

The method for modal block clustering is as follows.

1. Create a list of classes and subclasses.
2. Create a list of class attributes (e.g., methods, instance variables, system toolbox calls, physical devices manipulated).
3. Create a matrix of the class and subclasses by attributes.
4. Rate all attributes for all classes and subclasses in a binary fashion (e.g., the XX class uses the DragGrayRgn() toolbox call so the DragGrayRgn() attribute is assigned a 1).
5. Perform a modal block cluster analysis on the matrix data.

A network graphical representation can then be created to represent the software library based on the modal block cluster analysis. Links in the network are defined by shared items (i.e., classes overlapping multiple blocks). Any blocks that share classes are then linked.

Programmers: Student Versus Professional

Very little research has been conducted to explicitly compare student and professional programmers. Many experiments that research programming issues, however, use student subjects. This has been a cause of concern for some individuals interested in software programming research. The question has been raised (e.g., Mayer, 1981) whether student programmers can truly be classified as expert because it is believed that student programmers have not had the time to create the necessary schemas that a professional programmer has had the time to develop.

If this supposition were true, then much of the existing literature is of questionable value for anything other than understanding novice performance in programming tasks. The situation, however, is not as clear cut as this line of argument indicates.

For example, Waddington and Henry (1990) conducted a study using professional telecommunications software experts. These "experts" varied in experience from 1 to 17 years of programming with a mean of 5.8 years. Another study by Jorgensen (1987) used "expert" student programmers with B.S. and M.S. degrees in computer science. Although no years of experience were listed, it is probably a safe assumption that these student programmers had at

least six years of experience (i.e., four years for a B.S. degree and two years for an M.S. degree).

One can ask whether these two groups were actually different from an experience perspective if the main measure of experience is the number of years spent programming. One could conjecture that probably a better measure of "expertise" is the number of years of programming and possibly the number and types of classes taken.

Two studies that have explicitly compared student and professional programmers were conducted by Boehm-Davis, Holt, Schultz, and Stanley (1986) and Holt, Boehm-Davis, and Schultz (1987). Boehm-Davis et al. (1986) studied the modifiability of code. The student programmers had less than 1 year of professional programming experience with a mean of 0.2 year. The professional programmers had from 1 to 12 years of professional programming experience with a mean of 3.5 years.

The main finding relevant for student versus professional programmers was that program structure had less of an effect on professional programmers than it did on student programmers for the modification task. The conclusion was that because professional programmers had more experience, this allowed them to look beyond the surface features of a program and to concentrate on the important data structures and processes.

The second study (Holt, Boehm-Davis, and Schultz, 1987) comparing student and professional programmers also examined modification difficulty. The only specification for the programmers was that student programmers were upper-division computer science majors. No mention is made of the number of years of experience or whether the "professional" programmers obtained through advertising were not actually students from another university.

The results from the study were that the mental models developed by students were affected by program structure and content whereas the mental models of professional programmers were only affected by the difficulty of the modification task. Thus, as in the previous study, students were more affected by surface features or structures than were professional programmers.

These findings are of interest to the current research because a case could be made that manipulating the structure of a software library is intertwined with changing surface structures. Clearly, a certain amount of the restructuring is based on things that are not surface features, such as methods and interconnectivity. However, the potential for an interesting interaction exists and therefore manipulation of this variable has been included in the current research.

Summary of Introduction

Mental models can be used to represent user models. Potentially, mental models could be captured and used to structure software libraries to ease the code location problem.

Software metrics can be used to capture aspects of software for experimentation. This should result in more control and stability in the experiment. The DIT and RFC metrics are useful for dealing with and understanding class/subclass trees.

Finally, student programmers are often used as subjects in programming studies. Rarely, however, have students been compared to professional programmers. This issue has led to many people in the area of software research to ask whether students are "real" expert programmers.

In two studies (Boehm-Davis et al., 1986; Holt et al., 1987), students were shown to be more susceptible to surface structure disruption in a program modification task than were professional programmers. This result makes the student versus professional programmer variable an important area of study for software library arrangement. It is conceivable that data obtained from student subjects are totally inappropriate for usage with professional programmers.

Research Objectives

Following are the objectives of this research:

I. To determine if mental model methods can be used to create better class library structures (i.e., can the library structure be better arranged to ease the code locating problem). Two of the more promising and efficient methods were chosen (i.e., hierarchical cluster analysis of sorting data and modal block clustering of rating data) to be compared to traditional class/subclass trees. Both of the new organization methods have been successfully applied previously.

II. To determine if two object-oriented software metrics (i.e., DIT and RFC) are relevant to library locating measures such as RT and error rates. Both of these metrics have shown some success in a previous study for software maintenance (Lei, 1991; Lei, 1993).

III. To examine the use of student programmers as subjects in studies on code locating performance. There is a need to assess a common practice in software research (i.e., the use of student subjects) that is being questioned by many people, especially in industry.

METHOD

Experimental Design

A four-factor, mixed model factorial design was used. One factor was whether the subjects are student or professional programmers. Another factor was the type of representation used for the SPS User Interface Management System (UIMS) software library (i.e., hierarchical cluster analysis tree, class/subclass tree, or modal block cluster analysis network). The final factors were the class items classified by the Depth-In-Inheritance-Tree (i.e., DIT: 0-1, 2, 3-4) and Response-For-Class (i.e., RFC: 2-25, 26-118) software metrics. Graphically, the design is shown in Figures 3 and 4.

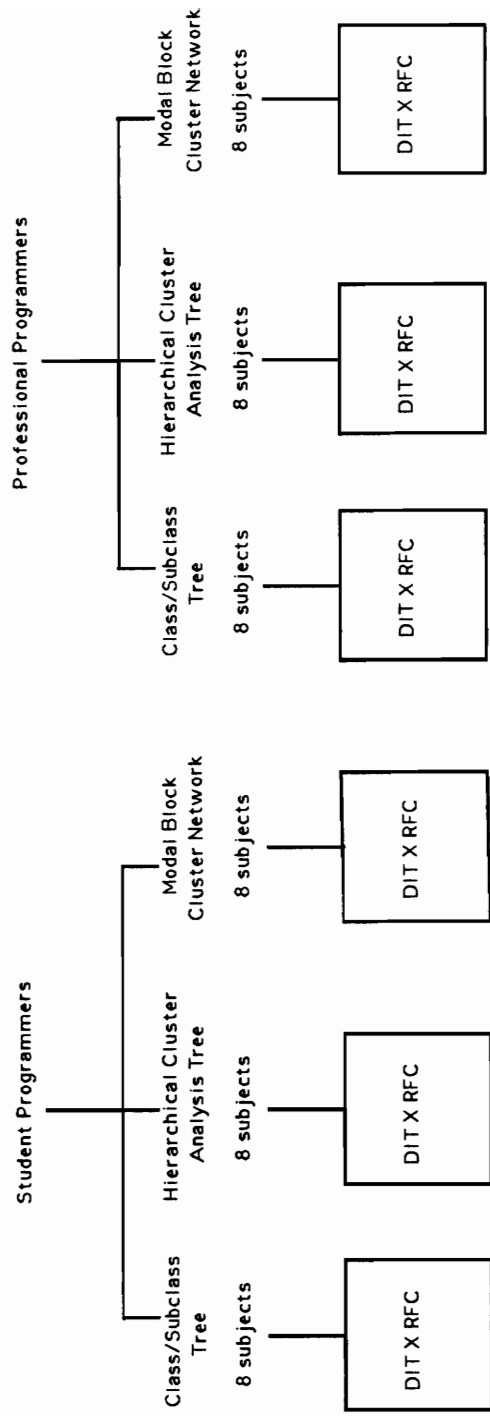


Figure 3. Top level between-subjects portion of the mixed design.

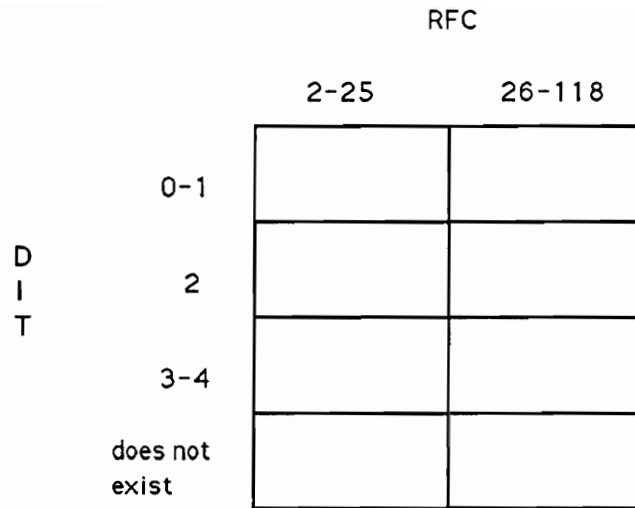


Figure 4. Diagram of the DIT X RFC part of the design seen by all subjects.

The experience/professional level of the programmers (2) and the type of representation (3) are between-subject factors. The DIT (2) X RFC (4) factor combination was seen by all the subjects within their particular programmer X representation conditions; thus, it is the within-subjects portion of the design.

The designs in Figures 3 and 4 apply directly to Task 1. Task 2 was similar except that the "does not exist" condition in the DIT X RFC portion of the design was not included.

Subjects

Two separate groups of subjects were used. The first group of 24 subjects was composed of senior computer science students taking a class in software engineering. The second group of 24 subjects was composed of professional programmers from Bell Northern Research. All 48 subjects completed the same tasks.

The student programmer subjects' general programming experience ranged from 2 to 12 years with a mean of 4.77 years. The student subjects' object-oriented experience ranged from 0 to 2 years with a mean of 0.11 year. Most had experience with the concepts of object-oriented programming but little actual programming experience. All of the students majored in computer science, computer engineering, or mathematics. All of the students had used Unix, PC, and Macintosh computers and all had also used command line and graphical user interfaces. Finally, all of the students used Pascal or C as their main programming language. Most of the students also had experience with another language (typically Basic, Fortran, C++, or Smalltalk).

The professional programmer subjects' general programming experience ranged from 2 to 14 years with a mean of 6.46 years. The professional subjects' object-oriented experience ranged from 0 to 4 years with a mean of 0.237 year. Most had experience with the concepts of object-oriented programming and a little actual

programming experience. The undergraduate and graduate majors of the professional programmers included computer science, computer engineering, electrical engineering, mathematics, and linguistics. All of the professional programmers had graduate degrees. All of the professional programmers had used Unix, PC, and Macintosh computers, as well as command line and graphical user interfaces. All of the professional programmers used the Protel[©] language on a regular basis. All of the professional programmers also had experience with C or Pascal. Most of the professional programmers also had experience with either Basic or Fortran. Finally, four of the professional programmers had extensive experience with Smalltalk or C++ and six more of the professional programmers had limited experience with Smalltalk or C++.

Materials and Apparatus

The SPS UIMS was used in the study. The original class/subclass tree was used as one representation. The data were also reorganized and presented in the manner outlined in the sections on hierarchical cluster analysis and modal block clustering.

The particular classes and subclasses that were to be located by the subjects in Task 1 are contained in Appendix A. The classes and subclasses that were to be located by the subjects in Task 2 are contained in Appendix B. Appendix C contains the complete list of classes and subclasses contained in UIMS and used in the representations for Tasks 1 and 2.

Classes and subclasses were selected from the UIMS library based on their metric ratings determined in a previous study (Lei, 1991, Lei, 1993). The particular metrics used were the Depth-in-Inheritance-Tree (DIT) metric and the Response-For-Class (RFC) metric (Chidamber and Kemerer, 1991).

The DIT metric is simply the location of a class within the original class/subclass tree in UIMS. The DIT metric ranges in values from 0 (base level class - no parent class) to 4 (i.e., 3 levels down from a base class). The three levels used in the study are 0 to 1, 2, and 3 to 4. The reason for the use of the DIT metric is that it also

relates to the visual structure of the class/subclass tree. A class is not only conceptually located in the inheritance tree, but is physically located in the inheritance tree for user interface display purposes.

The RFC metric is a count of all the local methods in a class and all the methods that the local methods call. The values of the RFC metric range from 2 to 156. The two levels used in the study are 2 to 25, and 26 to 118. The reason for the use of the RFC metric is that it is a simple, global measure that can be parsed out into a number of more focused metrics (e.g., LCOM - Lack of Cohesion of Class Metric, and WMC - Weighted Method per Class). A significant RFC effect would indicate that a more intense scrutiny of the classes with these related, but more specific metrics is merited.

Three different graphic representations (see bottom half of Figure 5 for an example) of the UIMS software library were developed for reuse. The first representation is the class/subclass tree. The second representation is the hierarchical cluster analysis tree based on similarity groupings. The third and final representation is the modal block cluster analysis network based on attribute ratings.

The software ran on a Macintosh Centris 610 computer with a monochrome display. Software to capture the mouse clicks was also

incorporated into the software library program to gather the times and responses of the subjects.

Procedure of Task 1

Time requirement. The task required about 45 minutes to complete.

Task sequence. Subjects were briefed as to their rights and then signed the subject rights statement. Subjects were informed that they would iterate through 48 trials (a trial is outlined below). Any questions by the subject were then answered.

The actual task sequence was as follows:

1. The trial was initiated with a 500-ms tone followed by a 1-s pause.
2. The name of the class (e.g., rectangle) that the subject was to find in the software library was then shown in the box at the top of the screen (see item 1 in Figure 5). All class names shown were presented in a random order between subjects.
3. The timer started when the word appeared in the box.
4. The subject then located the target class name in the displayed software library.

5. When the appropriate class name was located, the subject clicked on the class name.
6. The timer stopped when the subject released the mouse button.
7. If the subject believed that the class name he or she was searching for does not exist, then he or she clicked in the box in the upper left corner of the screen labeled as "class does not exist" (see item 2 in Figure 5).
8. The time to terminate a search was then recorded.

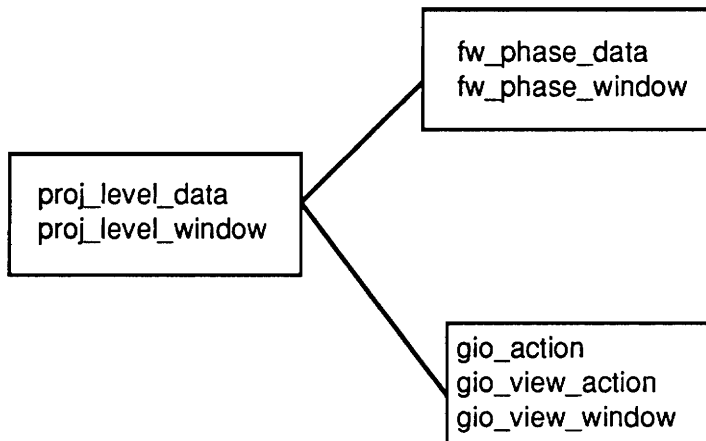
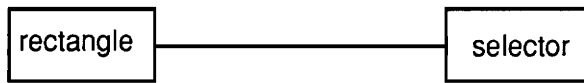
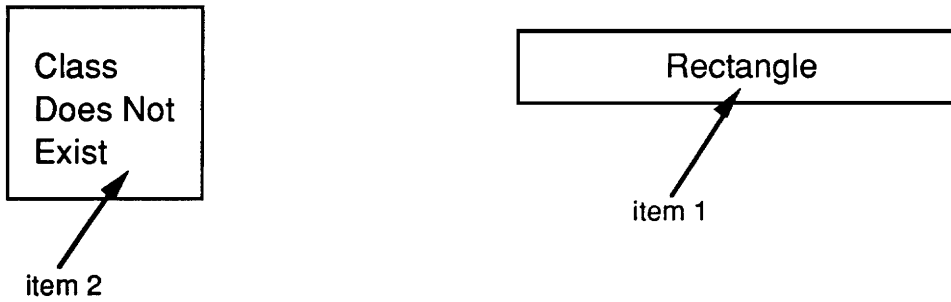


Figure 5. A representation of the computer screen for the tasks.

9. If the class name existed and the subject mistakenly clicked in the "class does not exist" box, then the decision to terminate the search was recorded as an error.

10. The class name box was then cleared and the trial was iterated.

Dependent variables. The first dependent variable of Task 1 was the time to find or time to terminate the search for a class. The second dependent variable was the error rate.

An error was counted when a class existed but the subject terminated the search by selecting the "class does not exist" box. An error was also counted when a class existed but the subject selected another class. Finally, the last addition to the error count was when a class did not exist and a subject selected some class anyway.

Procedure of Task 2

Time requirement. The task required about 45 minutes to complete.

Task sequence. The subject was briefed that he would need to determine if 30 specific requirements for a piece of software might be filled by using classes from the library used in Task 1. A goal statement and the specification requirements were presented to the subject on a sheet of paper. The goal statement and specification requirements are listed in Appendix D.

Thirty items (i.e., class names) existed in the library to satisfy each of the 30 specification requirements. When an item (i.e., class name) was found, the subject clicked on the class box (same as in

Task 1). The time from the start of the task to when the user clicked was recorded.

When an item was found, the subject was also asked to rate the class on how much of the class is usable on a 100-point scale from 0 to 10 (i.e., 0, .1, .2,..., 9.9, 10). The rating was used to determine if the representation (i.e., hierarchical cluster analysis tree, class/subclass tree, or modal block cluster network) affected how the user perceived the amount of code that could be reused in a class for the final project.

Finally, the time to make the rating was collected. The assumption behind this measurement was that a shorter time to make a rating decision indicates a higher degree of confidence. A slower time to rate may reflect a rating that is based on less conviction.

Dependent variables. The first dependent variable was the time to locate a class. The second dependent variable was the error rate. Errors in Task 2 were defined by a subject selecting an incorrect class. Note that in Task 2 there were no cases where the searched-for class did not exist. The other measures of interest were the rating of class reusability and the time to make the rating.

Experiment Conclusion

The subjects were debriefed and any questions were answered.

RESULTS

Task 1

Task 1 RT. An analysis of variance (ANOVA) was run on the response time (RT) data and the error data. The summary for the RT data is contained in Table 2. The within-subjects effects were corrected for violations of sphericity using the Greenhouse-Geiser (1959) correction.

For all statistical tests, the level of significance was set at $p \leq 0.05$. Also, where necessary, the post-hoc analyses were conducted using a Newman-Keuls paired test.

As shown in Table 2, the main effects of Representation [$F(2, 42) = 3.8, p = 0.03$], DIT [$F(3, 126) = 217.4, p_{G-G} = 0.0001$], and RFC [$F(1, 42) = 25.2, p = 0.0001$] are statistically significant. The post-hoc analysis on the Representation main effect indicates that the modal block cluster analysis organization is significantly worse (i.e., times to locate classes were slower) than the other two representations (i.e., class/subclass tree and hierarchical cluster analysis tree). The means of the three representations are contained in Table 3 and illustrated in Figure 6.

TABLE 2

ANOVA Summary Table for Task 1 Reaction Time

Source	df	Mean Square	F	p	η^2_{G-G}	ϵ_{G-G}
Programmer (P)	1	4.0115	.0198	.8887		
Representation (R)	2	768.9159	3.7969	.0305		
Programmer * R	2	4.7828	.0236	.9767		
Subject(Group) (S/G)	42	202.5133				
DIT	3	14379.0552	217.4398		.0001	.4769
DIT * P	3	174.2876	2.6356	.0527		
DIT * R	6	510.2620	7.7162		.0002	.4769
DIT * P * R	6	10.5760	.1599	.9867		
DIT * S/G	126	66.1289				
RFC	1	959.5848	25.1801	.0001		
RFC * P	1	159.7149	4.1910	.0469		
RFC * R	2	43.8391	1.1504	.3263		
RFC * P * R	2	16.7408	.4393	.6474		
RFC * S/G	42	38.1089				
DIT * RFC	3	733.7906	23.0849		.0001	.7607
DIT * RFC * P	3	5.9857	.1883	.9042		
DIT * RFC * R	6	46.8208	1.4730	.1926		
DIT * RFC * P * R	6	13.3621	.4204	.8643		
DIT * RFC * S/G	126	31.7865				
Total	383					

TABLE 3

Mean Response Times for the Representation Conditions, Task 1

Class/Subclass	Count	Mean	Std. Dev.	Std. Error
Class/Subclass	128	18.6340	11.2601	.9953
Hierarchical Cluster Analysis	128	19.3223	11.5585	1.0216
Modal Block Cluster	128	23.1813	17.7458	1.5685

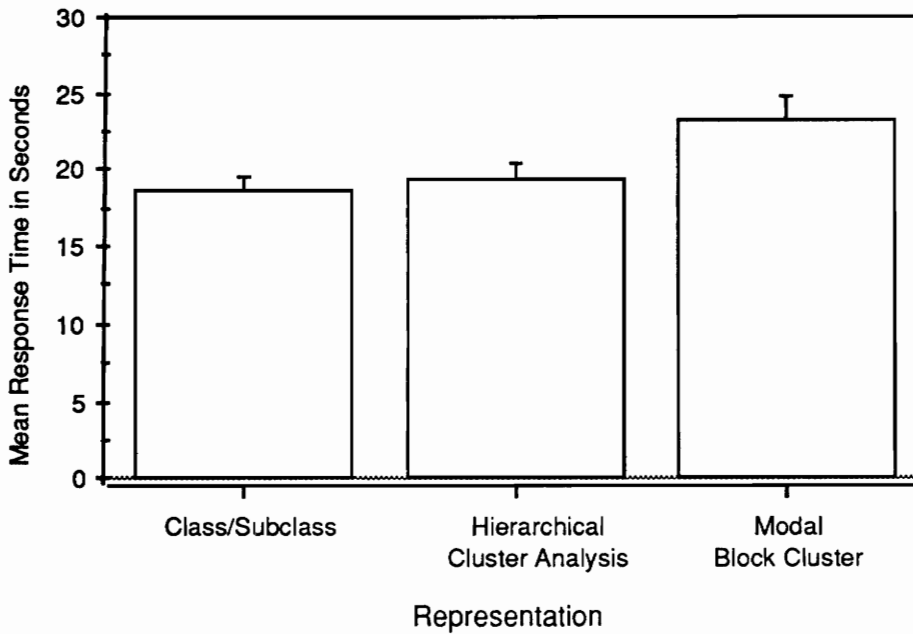


Figure 6. Mean response times of the representations with standard error bars.

A Newman-Keuls analysis on the significant DIT main effect indicates that all levels of DIT are significantly different from all the other DIT levels except for one. Only the DIT-2:DIT-3 comparison is not significant.

The means of the DIT levels (Table 4) show that DIT-2 (i.e., the second level in the original class/subclass tree) and DIT-3 (i.e., the third and fourth levels in the original class/subclass tree) are significantly slower than DIT-1 (i.e., the first level in the original class/subclass tree). DIT-4 (i.e., class does not exist) was more than twice as slow as DIT-2 and DIT-3 and almost 3.5 times slower than DIT-1.

In other words, classes in the middle of the tree or at the bottom (or end) of the tree required more time to be located than classes at the start (or top) of the tree. Classes that did not exist required the largest search times of all to determine that they, in fact, were not present in the class library. The means of the DIT levels are shown in Table 4 and illustrated in Figure 7.

TABLE 4

Mean Response Times for the DIT Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
DIT-1	96	11.2212	5.5628	.5677
DIT-2	96	15.9516	7.9715	.8136
DIT-3	96	15.9117	5.8951	.6017
DIT-4	96	38.4323	14.2111	1.4504

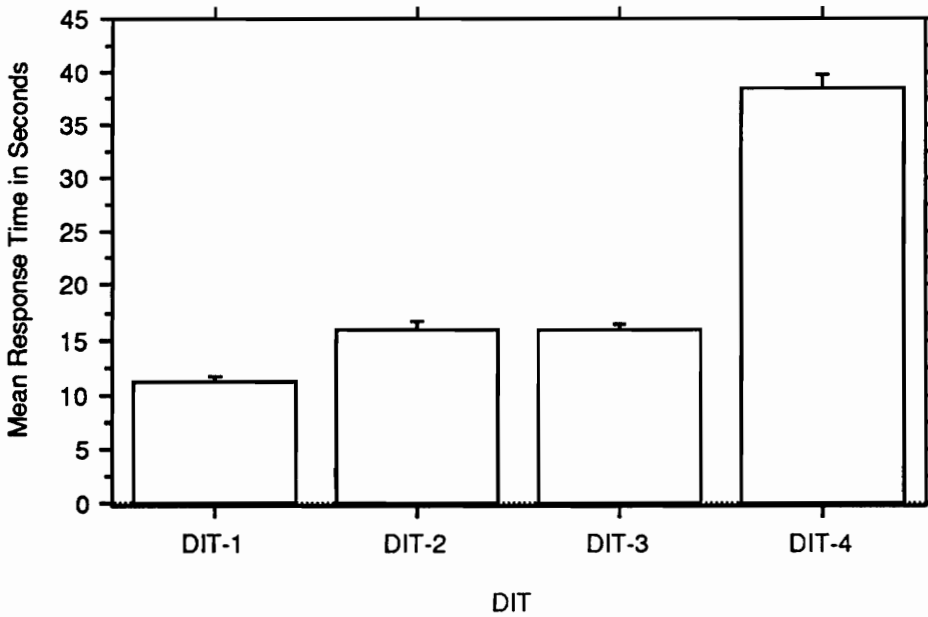


Figure 7. Mean response times of the DIT levels with standard error bars.

Classes in the RFC-2 levels (i.e., classes with more local methods and more calls by the local methods to other methods) take longer to locate than classes in the RFC-1 level. The means of the RFC levels are shown below in Table 5.

TABLE 5

Mean Response Times for the RFC Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
RFC-1	192	18.7984	15.0091	1.0832
RFC-2	192	21.9600	12.6603	.9137

Three two-way interactions are statistically significant. The DIT X Representation interaction had a p_{G-G} value of 0.0002 ($F(6, 126) = 7.7$, Table 2). The means of the DIT X Representation interaction are contained in Table 6 and plotted in Figure 8. A Newman-Keuls analysis (at $p \leq 0.05$) for the DIT X Representation interaction is summarized in Table 7.

TABLE 6

Mean Response Times for the DIT X Representation Interaction Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
DIT-1, C/Sc	32	12.0638	6.5840	1.1639
DIT-1, HCA	32	11.9523	5.3363	.9433
DIT-1, MBC	32	9.6476	4.3778	.7739
DIT-2, C/Sc	32	13.4062	7.9165	1.3995
DIT-2, HCA	32	15.6945	7.2306	1.2782
DIT-2, MBC	32	18.7540	8.0571	1.4243
DIT-3, C/Sc	32	15.8053	5.0236	.8881
DIT-3, HCA	32	15.1050	5.6990	1.0075
DIT-3, MBC	32	16.8247	6.8731	1.2150
DIT-4, C/Sc	32	33.2606	9.2897	1.6422
DIT-4, HCA	32	34.5375	10.3489	1.8295
DIT-4, MBC	32	47.4988	17.2446	3.0484

C/Sc : Class/Subclass
HCA: Hierarchical Cluster Analysis
MBC: Modal Block Cluster

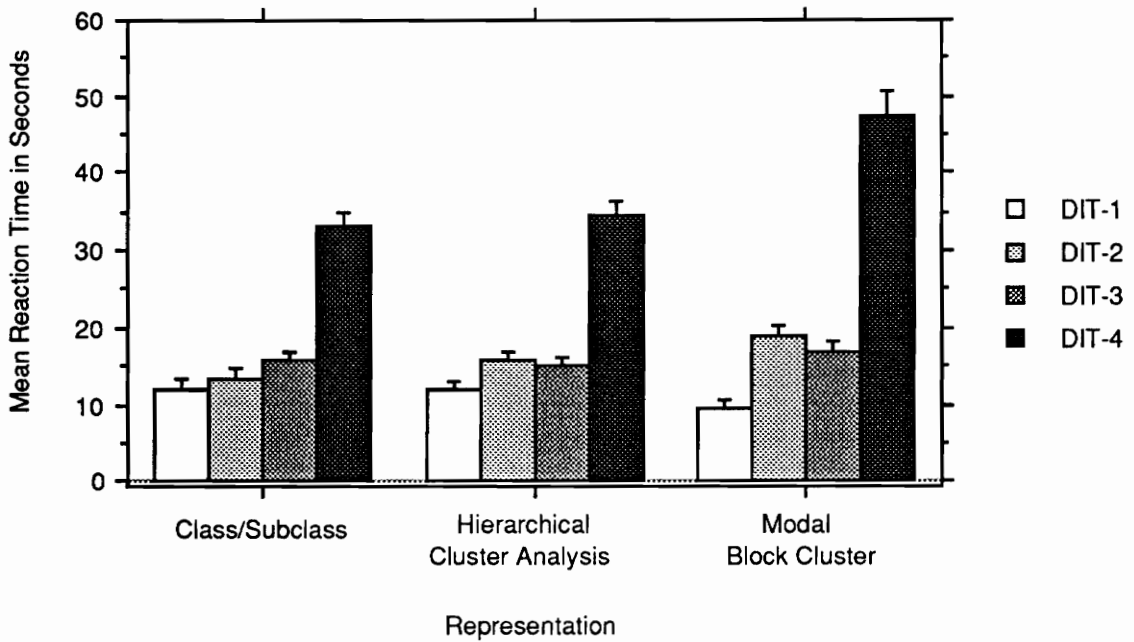


Figure 8. Mean response time of the DIT X Representation interaction with standard error bars.

TABLE 7

Newman-Keuls Post-Hoc Analysis for the DIT X Representation Interaction for Response Times, Task 1

	R3D1	R2D1	R1D1	R1D2	R2D3	R2D2	R1D3	R3D3	R3D2	R1D4	R2D4	R3D4
R3D1	- - -	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
R2D1		- - -	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
R1D1			- - -	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
R1D2				- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.
R2D3					- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.
R2D2						- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.
R1D3							- - -	not sig.	not sig.	not sig.	not sig.	Sig.
R3D3								- - -	not sig.	not sig.	not sig.	not sig.
R3D2									- - -	not sig.	not sig.	not sig.
R1D4										- - -	not sig.	not sig.
R2D4											- - -	not sig.

R: Representation

R1 = Representation - 1 (Class/Subclass Tree)

R2 = Representation - 2 (Hierarchical Cluster Analysis Tree)

R3 = Representation - 3 (Modal Block Clusters)

D: DIT

D1 = DIT-1 (first layer in class/subclass tree)

D2 = DIT-2 (second layer in class/subclass tree)

D3 = DIT-3 (third and fourth layers in class/subclass tree)

D4 = DIT-4 (class does not exist)

A number of the paired comparisons are significant as the Newman-Keuls analysis in Table 7 shows. The most obvious result is that $R3D4 = R2D4 = R1D4 >$ all other cases. Thus, for all three representations, the searches involving classes that did not exist proved to be much more time consuming for the subjects. It is also clear that $R3D1$ resulted in the shortest search times. Thus, the modal block cluster representation resulted in the fastest search times for classes from DIT-1.

Other findings from the Newman-Keuls analysis are the following: ($R3D2 > R1D2, R1D1, R2D1, R3D1$), ($R3D3 > R1D1, R2D1, R3D1$), ($R1D3 > R1D1, R2D1, R3D1$), ($R2D2 > R2D1, R3D1$), and ($R2D3 > R3D1$). Thus, it appears that the middle levels of DIT (i.e., DIT-2) and the deeper levels of DIT (i.e., DIT-3) for the modal block cluster representation are consistently slower than the shallow level (i.e., DIT-1) for the other two representations. The deepest level (i.e., DIT-3) for the class/subclass tree was also consistently slower than the most shallow DIT level (i.e., DIT-1) for all representations.

The RFC X Programmer interaction ($F(1, 42) = 25.2, p = 0.0469$) is also statistically significant. The means of the RFC X Programmer interaction are contained in Table 8 and are illustrated in Figure 9.

TABLE 8

Mean Response Times for the RFC X Programmer Interaction Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
RFC-1, Stu	96	18.0513	14.4061	1.4703
RFC-1, Pro	96	19.5455	15.6285	1.5951
RFC-2, Stu	96	22.5027	13.6457	1.3927
RFC-2, Pro	96	21.4173	11.6386	1.1879

Stu: Student
Pro: Professional

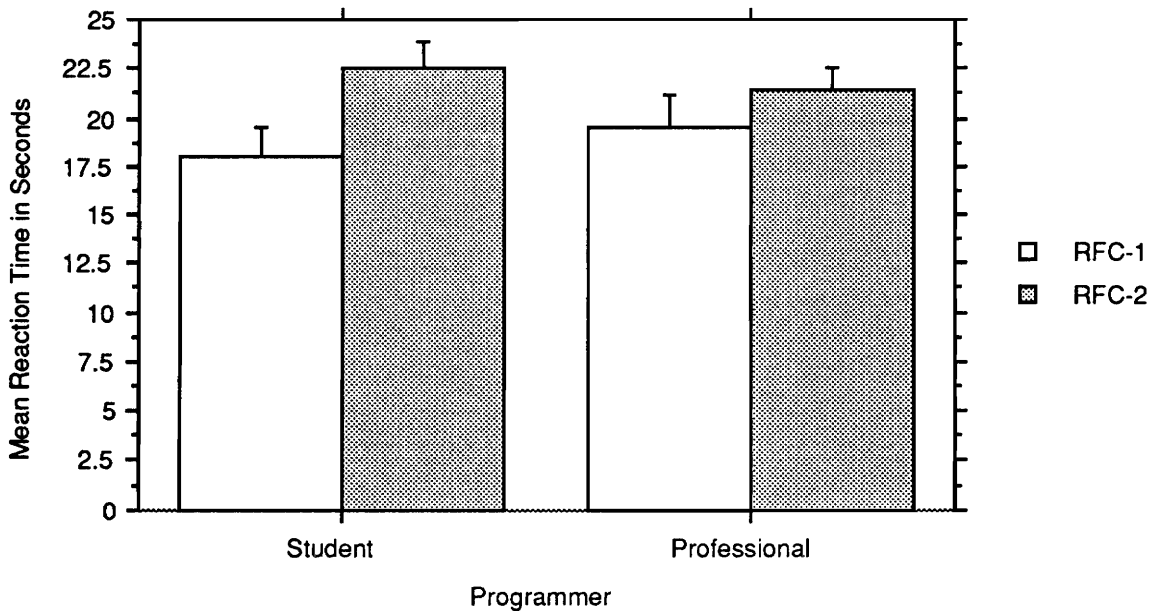


Figure 9. Mean response times of the RFC X Programmer interaction with standard error bars.

The Newman-Keuls analysis shows that only two of the comparisons are significantly different from one another (i.e., student/RFC-2 > student/RFC-1 and student/RFC-1 < professional/RFC-2). Interestingly, student programmers are faster when dealing with RFC-1 classes than when dealing with RFC-2 classes, whereas this difference did not exist for the professional programmers.

Finally, the DIT X RFC interaction ($F(3, 126) = 23, p_{G-G} = 0.0001$) is also statistically significant. The means of the DIT X RFC interaction are contained in Table 9 and plotted in Figure 10. A Newman-Keuls analysis is summarized in Table 10.

TABLE 9

Mean Response Times for the DIT X RFC Interaction Conditions,
Task 1

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1	48	9.2691	5.9403	.8574
DIT-1, RFC-2	48	13.1734	4.4133	.6370
DIT-2, RFC-1	48	10.6164	5.4746	.7902
DIT-2, RFC-2	48	21.2867	6.3507	.9166
DIT-3, RFC-1	48	16.6999	6.6864	.9651
DIT-3, RFC-2	48	15.1234	4.9261	.7110
DIT-4, RFC-1	48	38.6081	15.4635	2.2320
DIT-4, RFC-2	48	38.2565	13.0009	1.8765

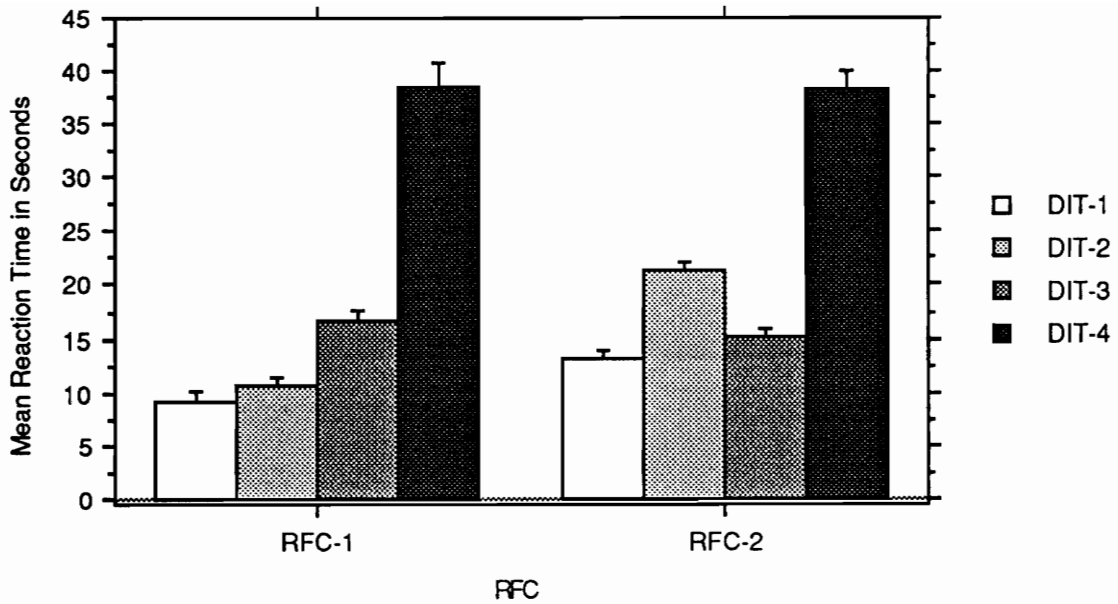


Figure 10. Mean response times for the DIT X RFC interaction with standard error bars.

The Newman-Keuls analysis indicates that the cases where the classes did not exist basically resulted in the longest search times (DIT-4/RFC-2 = DIT-4/RFC-1 = DIT-2/RFC-2 > remaining cases). Another finding was that the classes in DIT-2/RFC-2 required more time to locate than did the classes in DIT-3/RFC-2 and DIT-3/RFC-1. Also, classes in DIT-3/RFC-1 required more time to locate than classes in DIT-2/RFC-1 and DIT-1/RFC-1.

Finally, for DIT-1 and DIT-2, the RFC-2 classes took more time to locate than did the RFC-1 classes. This was not true for DIT-3 and DIT-4.

TABLE 10

Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Response Times, Task 1

	DIT-1 RFC-1	DIT-2 RFC-1	DIT-1 RFC-2	DIT-3 RFC-2	DIT-3 RFC-1	DIT-2 RFC-2	DIT-4 RFC-2	DIT-4 RFC-1
DIT-1 RFC-1	- - - -	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
DIT-2 RFC-1		- - - -	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.
DIT-1 RFC-2			- - - -	not sig.	Sig.	Sig.	Sig.	Sig.
DIT-3 RFC-2				- - - -	not sig.	Sig.	Sig.	Sig.
DIT-3 RFC-1					- - - -	not sig.	Sig.	Sig.
DIT-2 RFC-2						- - - -	not sig.	Sig.
DIT-4 RFC-2							- - - -	not sig.

Task 1 Errors. The errors that occurred during class location (i.e., the wrong class was selected by the subject) were also recorded and analyzed with an ANOVA. The results are summarized in Table 11.

TABLE 11

ANOVA Summary Table for Task 1 Error Rate

Source	df	Mean Square	F	p	p_{G-G}	ϵ_{G-G}
Programmer (P)	1	.5104	1.2120	.2772		
Representation (R)	2	1.5729	3.7350	.0321		
P * R	2	.9479	2.2509	.1179		
Subject(Group) (S/G)	42	.4211				
DIT	3	19.1562	59.1409		.0001	.9136
DIT * P	3	.2812	.8683	.4595		
DIT * R	6	.6563	2.0260	.0669		
DIT * P * R	6	.4479	1.3828	.2265		
DIT * S/G	126	.3239				
RFC	1	10.0104	30.1659	.0001		
RFC * P	1	.0937	.2825	.5979		
RFC * R	2	1.3229	3.9865	.0260		
RFC * P * R	2	.2812	.8475	.4357		
RFC * S/G	42	.3318				
DIT * RFC	3	23.8785	67.1395		.0001	.8673
DIT * RFC * P	3	.5035	1.4156	.2413		
DIT * RFC * R	6	.4826	1.3570	.2371		
DIT * RFC * P * R	6	.1493	.4198	.8647		
DIT * RFC * S/G	126	.3557				
Total	383					

The main effects of Representation ($F(2, 42) = 3.7, p = 0.03$), DIT ($F(3, 126) = 59.1, p_{G-G} = 0.0001$), and RFC ($F(1, 42) = 30.2, p = 0.0001$) are all statistically significant. The means for Representation are shown in Table 12 and illustrated in Figure 11.

TABLE 12

Mean Error Rates for the Representation Conditions, Task 1

Class/Subclass	Count	Mean	Std. Dev.	Std. Error
Hierarchical Cluster Analysis	128	.7812	.9216	.0815
Modal Block Cluster	128	.6406	.8670	.0766
Class/Subclass	128	.5625	.7503	.0663

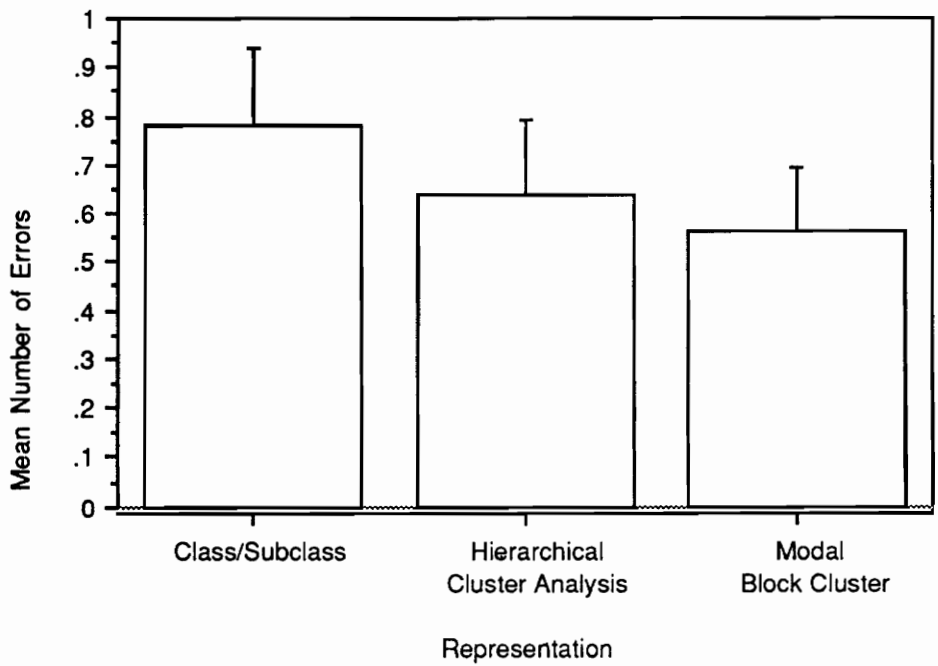


Figure 11. Mean errors of the Representations with standard error bars.

The results from the Newman-Keuls analysis indicate that the only comparison that is significantly different is that the class/subclass tree has a higher error rate than the modal block cluster.

The means for the significant DIT main effect are shown in Table 13 and plotted in Figure 12.

TABLE 13

Mean Error Rates for the DIT Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
DIT-1	96	.3958	.6237	.0637
DIT-2	96	.9583	1.0353	.1057
DIT-3	96	1.1146	.8319	.0849
DIT-4	96	.1771	.4103	.0419

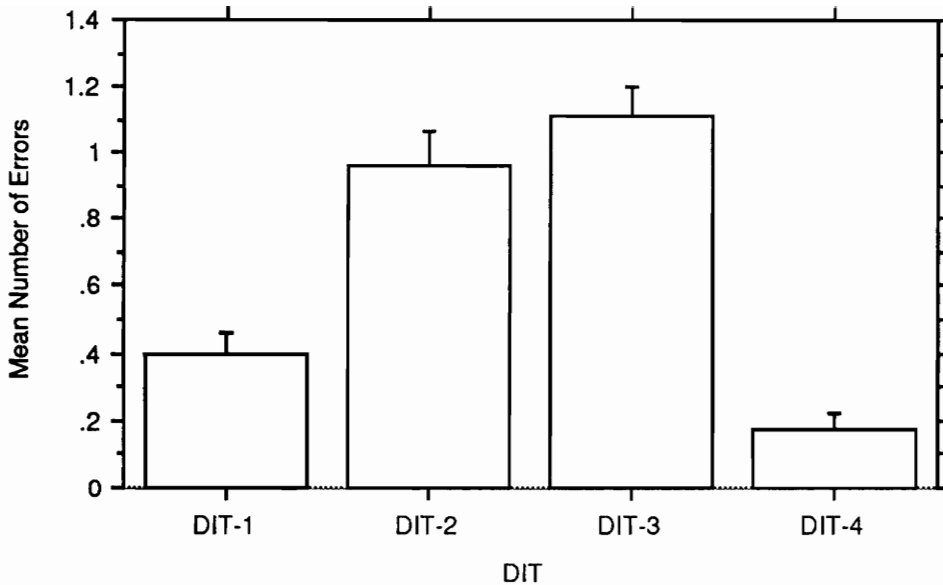


Figure 12. Mean errors for the DIT with standard error bars.

The Newman-Keuls analysis shows that all of the means are significantly different from one another. Error rates rose as the Depth-in-Inheritance-Tree increased for classes (i.e., DIT-3 error rates > DIT-2 error rates > DIT-1 error rates). The lowest error rates occurred for the DIT-4 level (i.e., the class being searched for does not exist).

RFC-2 classes resulted in more errors than RFC-1 classes for the search task in Task 1. The means for the RFC main effect are contained in Table 14.

TABLE 14

Mean Error Rates for the RFC Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
RFC-1	192	.5000	.7520	.0543
RFC-2	192	.8229	.9152	.0660

There are also two statistically significant two-way interactions: RFC X Representation [$F(2, 42) = 3.99, p_{G-G} = 0.026$], and DIT X RFC [$F(3, 126) = 67.1, p_{G-G} = 0.0001$]. The means for the RFC X Representation interaction are contained in Table 15 and plotted in Figure 13.

TABLE 15

Mean Error Rates for the RFC X Representation Interaction Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
RFC-1, C/Sc	64	.5156	.8163	.1020
RFC-1, HCA	64	.5781	.7929	.0991
RFC-1, MBC	64	.4062	.6354	.0794
RFC-2, C/Sc	64	1.0469	.9500	.1188
RFC-2, HCA	64	.7031	.9374	.1172
RFC-2, MBC	64	.7188	.8256	.1032

C/Sc: Class/Subclass
HCA: Hierarchical Cluster Analysis
MBC: Modal Block Cluster

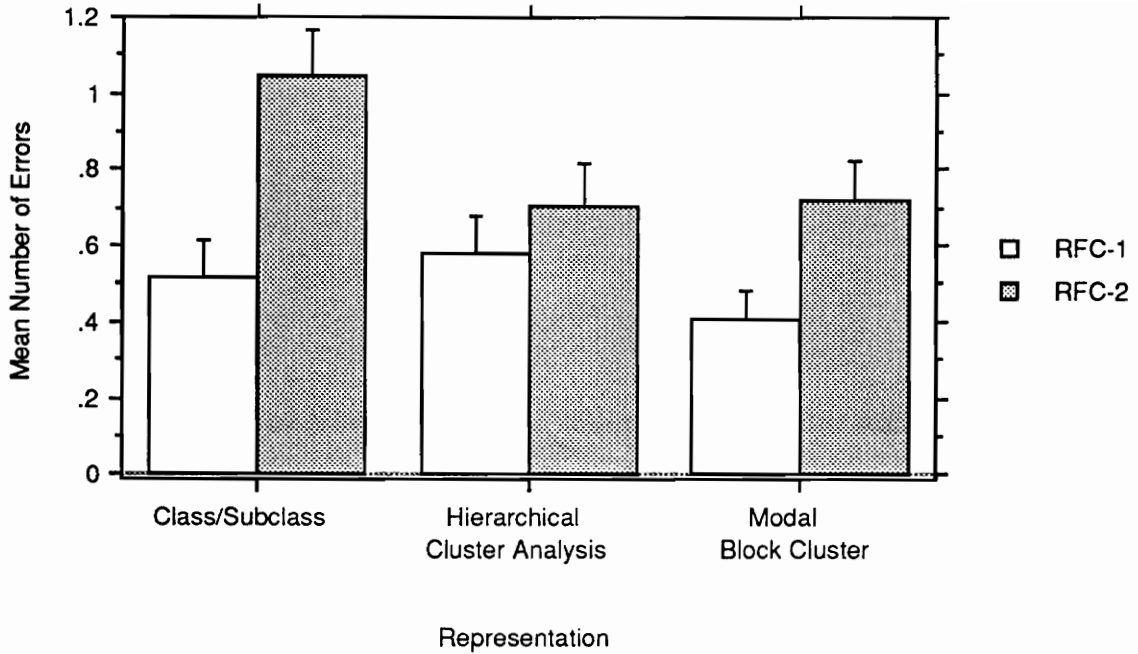


Figure 13. Mean errors for the RFC X Representation interaction with standard error bars.

The Newman-Keuls analysis is summarized in Table 16. As the table shows, the RFC-2/Class-Subclass Tree resulted in significantly more errors than any of the RFC-1/representation conditions. Two other comparisons are also significant. The RFC-2/Hierarchical Cluster Analysis condition had more errors than the RFC-1/Modal Block Cluster, and also the RFC-1 classes resulted in less errors than the RFC-2 classes for the Modal Block Cluster.

Thus, the arrangement that resulted in the most errors, in this study, was when a Class/Subclass Tree is used for RFC-2 classes. This

arrangement, however, was only significantly different from the RFC-1 classes. The other two representations were not significantly different on the number of errors for RFC-2 (between themselves).

The Modal Block Cluster used for RFC-1 classes had fewer errors than the RFC-2 classes for all representations but it did not have significantly fewer errors than the RFC-1 classes for either the Class/Subclass Tree or the Hierarchical Cluster Analysis Tree.

TABLE 16

Newman-Keuls Post-Hoc Analysis for the RFC X Representation Interaction for Error Rates, Task 1

	RFC-1: MBC	RFC-1: C/SC	RFC-1: HCA	RFC-2: HCA	RFC-2: MBC	RFC-2: C/SC
RFC-1: MBC	- - - -	not sig.	not sig.	Sig.	Sig.	Sig.
RFC-1: C/SC		- - - -	not sig.	not sig.	not sig.	Sig.
RFC-1: HCA			- - - -	not sig.	not sig.	Sig.
RFC-2: HCA				- - - -	not sig.	not sig.
RFC-2: MBC					- - - -	not sig.

Representation:

C/SC: Class/Subclass Tree

HCA: Hierarchical Cluster Analysis Tree

MBC: Modal Block Cluster

The means for the DIT X RFC interaction are contained in Table 17 and illustrated in Figure 14. A Newman-Keuls analysis is summarized in Table 18.

TABLE 17

Mean Error Rates for the DIT X RFC Interaction Conditions, Task 1

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1	48	.4167	.6469	.0934
DIT-1, RFC-2	48	.3750	.6058	.0874
DIT-2, RFC-1	48	.0833	.2793	.0403
DIT-2, RFC-2	48	1.8333	.7244	.1046
DIT-3, RFC-1	48	1.3958	.7363	.1063
DIT-3, RFC-2	48	.8333	.8337	.1203
DIT-4, RFC-1	48	.1042	.3087	.0446
DIT-4, RFC-2	48	.2500	.4838	.0698

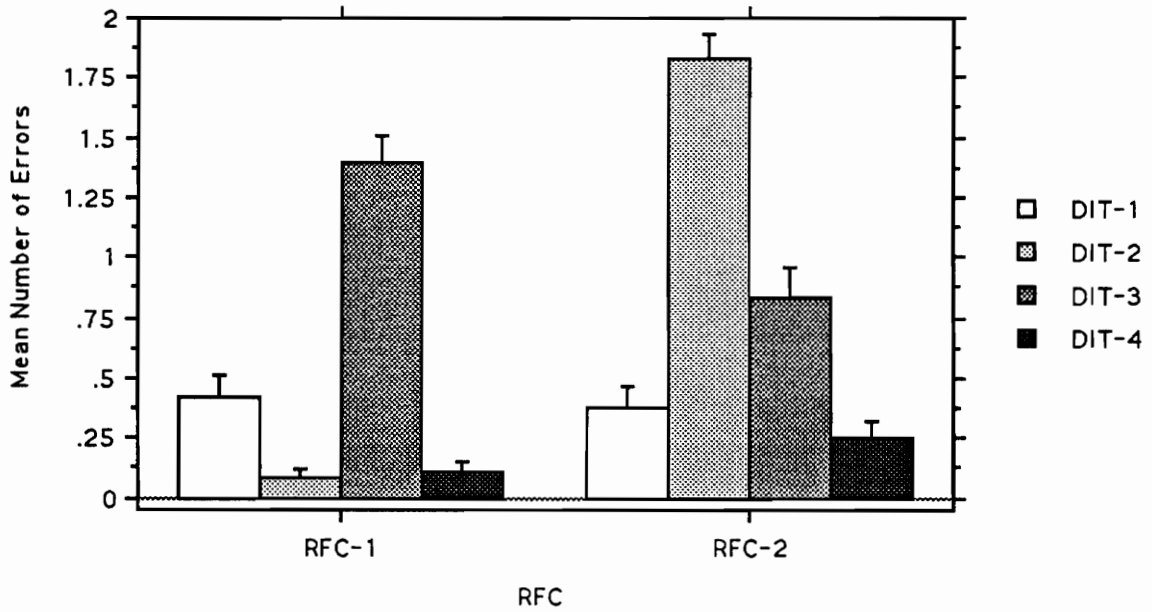


Figure 14. Mean errors for the DIT X RFC interaction with standard error bars.

The Newman-Keuls analysis shows that a number of the paired comparisons are significantly different from one another. A few exceptions are the following: DIT-1/RFC-1 versus DIT-1/RFC-2 or DIT-4/RFC-2, DIT-1/RFC-2 versus DIT-4/RFC-2 or DIT-4/RFC-1, and DIT-4/RFC-1 versus DIT-2/RFC-1.

The worst cases from the Newman-Keuls analysis for error rates on the DIT X RFC interaction are the DIT-2/RFC-2 = (DIT-3/RFC-1 and DIT-3/RFC2) > remaining cases. Interestingly, the DIT-1/RFC-1 also had a higher error rate than the DIT-4/RFC-1 and DIT-2/RFC-1.

TABLE 18

Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Error Rates, Task 1

	DIT-2 RFC-1	DIT-4 RFC-1	DIT-4 RFC-2	DIT-1 RFC-2	DIT-1 RFC-1	DIT-3 RFC-2	DIT-3 RFC-1	DIT-2 RFC-2
DIT-2 RFC-1	- - - -	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.
DIT-4 RFC-1		- - - -	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.
DIT-4 RFC-2			- - - -	not sig.	not sig.	Sig.	Sig.	Sig.
DIT-1 RFC-2				- - - -	not sig.	not sig.	Sig.	Sig.
DIT-1 RFC-1					- - - -	not sig.	not sig.	Sig.
DIT-3 RFC-2						- - - -	not sig.	not sig.
DIT-3 RFC-1							- - - -	not sig.

Task 2

Task 2 RT. The only main effect that is statistically significant is Programmer ($F(1, 28) = 4.2, p = 0.05$). The complete ANOVA table is shown in Table 19. The means for the Programmer main effect are shown in Table 20. As Table 20 shows, professional programmers took longer to locate the desired classes than did student programmers.

TABLE 19

ANOVA Summary Table for Task 2 Reaction Time

Source	df	Mean Square	F	p	p_{G-G}	ϵ_{G-G}
Programmer (P)	1	3956.3258	4.1958	.0500		
Representation (R)	2	1112.3240	1.1796	.3222		
P * R	2	41.5762	.0441	.9569		
Subject(Group) (S/G)	28	942.9339				
DIT	2	203.4333	.4485	.6408		
DIT * P	2	46.1117	.1017	.9035		
DIT * R	4	587.8953	1.2962	.2826		
DIT * P * R	4	842.4987	1.8575	.1307		
DIT * S/G	56	453.5568				
RFC	1	49.7991	.1128	.7394		
RFC * P	1	179.7416	.4073	.5285		
RFC * R	2	897.6035	2.0338	.1497		
RFC * P * R	2	593.2348	1.3442	.2771		
RFC * S/G	28	441.3405				
DIT * RFC	2	10173.2184	32.3584		.0001	.9166
DIT * RFC * P	2	920.9044	2.9292	.0617		
DIT * RFC * R	4	450.0358	1.4315	.2356		
DIT * RFC * P * R	4	121.6113	.3868	.8172		
DIT * RFC * S/G	56	314.3915				
Total	203					

TABLE 20

Mean Response Times for the Programmer Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
Student	102	32.2724	20.2720	2.0072
Professional	102	41.4056	27.5838	2.7312

The only two-way interaction that is statistically significant is the DIT X RFC interaction ($F(2, 56) = 32.4, p_{G-G} = 0.0001$). The means for the interaction are shown in Table 21 and the means are graphed in Figure 15.

TABLE 21

Mean Response Times for the DIT X RFC Interaction Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1	34	44.5294	26.3630	4.5212
DIT-1, RFC-2	34	27.5941	12.7804	2.1918
DIT-2, RFC-1	34	23.6638	9.4949	1.6284
DIT-2, RFC-2	34	53.4935	36.4874	6.2575
DIT-3, RFC-1	34	41.2000	22.9341	3.9332
DIT-3, RFC-2	34	30.5529	15.3515	2.6328

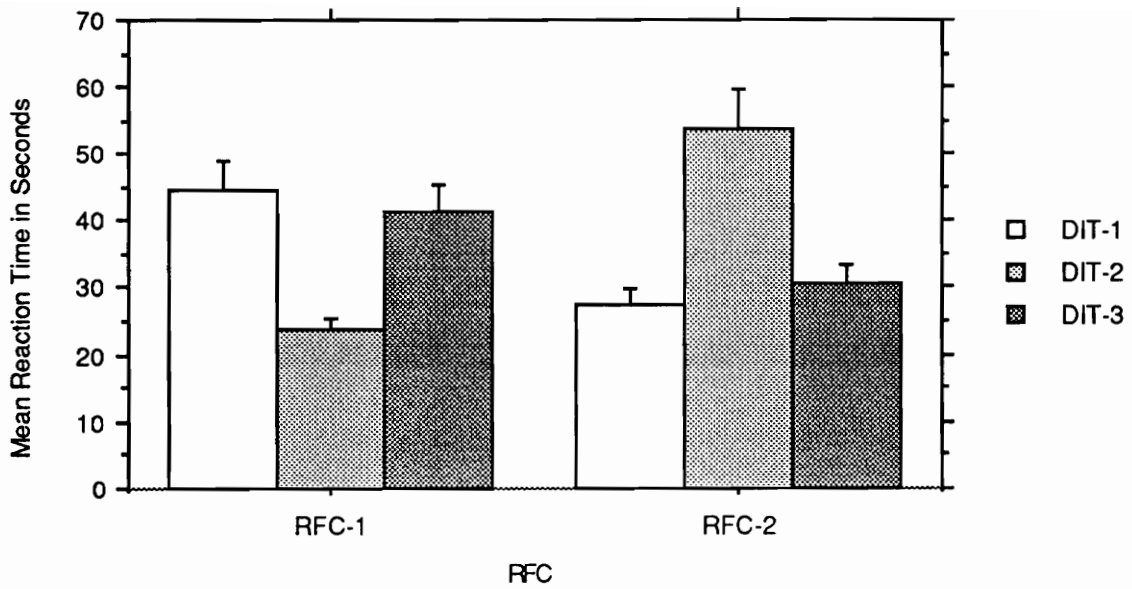


Figure 15. Mean response times for the DIT X RFC interaction with standard error bars.

A Newman-Keuls test indicates that most of the DIT X RFC conditions are significantly different from the others, except for the DIT-2/RFC-2 versus DIT-1/RFC-1, DIT-1/RFC-1 versus DIT-3/RFC-1 and DIT-3/RFC-2 versus DIT-1/RFC-2 comparisons. Of particular interest is that the DIT-2/RFC-1 condition is located fastest of all the conditions but the DIT-2/RFC-2 is the slowest of all the conditions (DIT-2/RFC-2 is significantly slower than all conditions except for DIT-1/RFC-1). Also of note is that DIT-1/RFC-1 is located more

slowly than DIT-1/RFC-2 and DIT-3/RFC-2. The results are shown in Table 22.

TABLE 22

Newman-Keuls Post-Hoc Analysis for the DIT X RFC Interaction for Response Times, Task 2

	DIT-2 RFC-1	DIT-1 RFC-2	DIT-3 RFC-2	DIT-3 RFC-1	DIT-1 RFC-1	DIT-2 RFC-2
DIT-2 RFC-1	- - - -	Sig.	Sig.	Sig.	Sig.	Sig.
DIT-1 RFC-2		- - - -	not sig.	Sig.	Sig.	Sig.
DIT-3 RFC-2			- - - -	Sig.	Sig.	Sig.
DIT-3 RFC-1				- - - -	not sig.	Sig.
DIT-1 RFC-1					- - - -	not sig.

Task 2 Ratings. An ANOVA (Table 23) conducted on the ratings obtained showed no statistically significant differences except for RFC ($F(1, 28) = 8.1, p = 0.008$). The means for RFC are shown in Table 24. The ratings were collected to determine if different factors could affect the perception of a class's reusability. The means indicate that the RFC-1 class was judged by the subjects to be more reusable.

TABLE 23

ANOVA Summary Table for the Task 2 Ratings

Source	df	Mean Square	F	p
Programmer (P)	1	890.9669	1.4958	.2315
Representation (R)	2	1084.8725	1.8213	.1805
P * R	2	691.2517	1.1605	.3279
Subject(Group) (S/G)	28	595.6521		
DIT	2	512.1499	2.3744	.1024
DIT * P	2	179.9816	.8344	.4395
DIT * R	4	116.5448	.5403	.7067
DIT * P * R	4	169.8401	.7874	.5383
DIT * S/G	56	215.7008		
RFC	1	1561.8457	8.1469	.0080
RFC * P	1	205.3262	1.0710	.3096
RFC * R	2	438.6798	2.2882	.1201
RFC * P * R	2	201.0744	1.0488	.3637
RFC * S/G	28	191.7114		
DIT * RFC	2	252.8147	1.2646	.2903
DIT * RFC * P	2	70.3628	.3520	.7048
DIT * RFC * R	4	250.0022	1.2505	.3003
DIT * RFC * P * R	4	142.1938	.7113	.5877
DIT * RFC * S/G	56	199.9144		
Total	203			

TABLE 24

Mean Ratings for RFC Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
RFC-1	102	77.7167	12.9770	1.2849
RFC-2	102	71.8020	19.6465	1.9453

Task 2 Rating RT. The complete results of the ANOVA are contained in Table 25. The time for the subject to make a rating was collected because it was believed that a rapid response carries more conviction on the part of the rater than does a slow response (which may indicate uncertainty).

The main effects of DIT ($F(2, 56) = 24.6, p_{G-G} = 0.0001$) and RFC ($F(1, 28) = 18.3, p = 0.0002$) were found to be statistically significant. The means for DIT and RFC are contained in Tables 26 and 27, respectively. The means for DIT are plotted in Figure 16.

TABLE 25

ANOVA Summary Table for the Rating Times

Source	df	Mean Square	F	p	p_{G-G}	ϵ_{G-G}
Programmer (P)	1	.4238	.0220	.8831		
Representation (R)	2	3.2757	.1701	.8444		
P * R	2	5.0724	.2634	.7703		
Subject(Group) (S/G)	28	19.2543				
DIT	2	328.5284	24.5675		.0001	.6245
DIT * P	2	13.0423	.9753	.3834		
DIT * R	4	23.3022	1.7425	.1535		
DIT * P * R	4	37.7274	2.8213		.0621	.6245
DIT * S/G	56	13.3725				
RFC	1	262.6290	18.2644	.0002		
RFC * P	1	10.8544	.7549	.3923		
RFC * R	2	18.0585	1.2559	.3004		
RFC * P * R	2	22.1264	1.5388	.2323		
RFC * S/G	28	14.3793				
DIT * RFC	2	414.1452	31.1292		.0001	.6776
DIT * RFC * P	2	4.0012	.3008	.7415		
DIT * RFC * R	4	14.4234	1.0841	.3732		
DIT * RFC * P * R	4	7.0990	.5336	.7116		
DIT * RFC * S/G	56	13.3041				
Total	203					

TABLE 26

Mean Rating Times for the DIT Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
DIT-1	68	3.1106	3.0947	.3753
DIT-2	68	6.7494	6.7397	.8173
DIT-3	68	2.9393	2.0998	.2546

TABLE 27

Mean Rating Times for the RFC Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
RFC-1	102	3.1417	2.6956	.2669
RFC-2	102	5.3912	5.9854	.5926

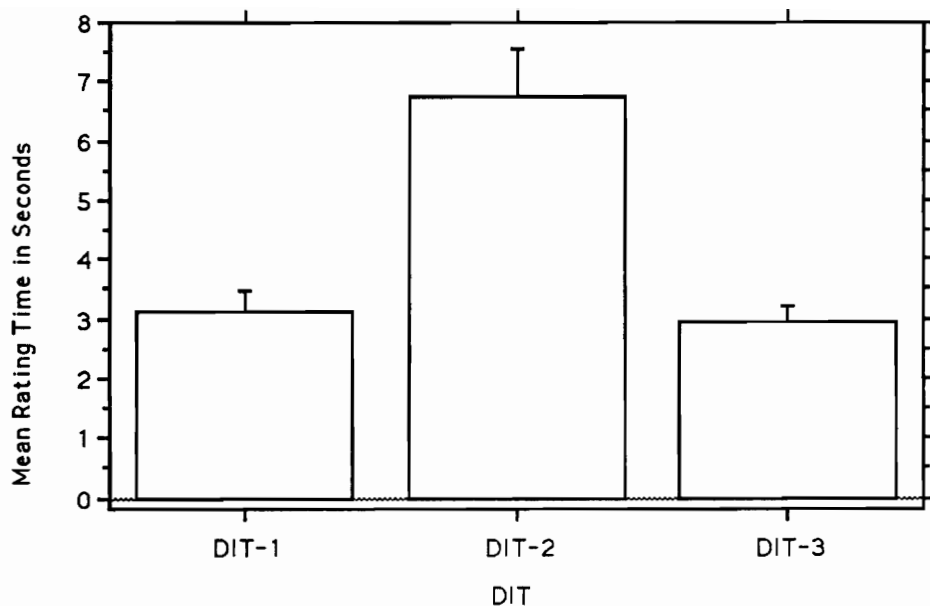


Figure 16. Mean rating times for the DIT with standard error bars.

The significant difference found for the DIT variable was further explored with a Newman-Keuls analysis. The results indicate that the time to make a rating decision for the DIT-2 class was greater than that for either the DIT-1 or DIT-3 class. The DIT-1 and DIT-3 levels were not significantly different.

The RFC-2 level required more time by the subjects to make a decision about the rating value than the RFC-1 level. This may be the most relevant analysis item for the Task 2 Rating-RT section because none of the following analysis items were correspondingly significant in the Task 2 RT analysis.

The DIT X RFC two-way effect was also found to be statistically significant ($F(2, 56) = 31.1, p_{G-G} = 0.0001$). The means for the DIT X RFC interaction are contained in Table 28 and are graphed in Figure 17.

TABLE 28

Mean Rating Times for the DIT X RFC Interaction Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1	34	3.9950	4.1052	.7040
DIT-1, RFC-2	34	2.2262	.9896	.1697
DIT-2, RFC-1	34	2.8747	1.3628	.2337
DIT-2, RFC-2	34	10.6241	7.7091	1.3221
DIT-3, RFC-1	34	2.5553	1.5333	.2630
DIT-3, RFC-2	34	3.3232	2.5093	.4303

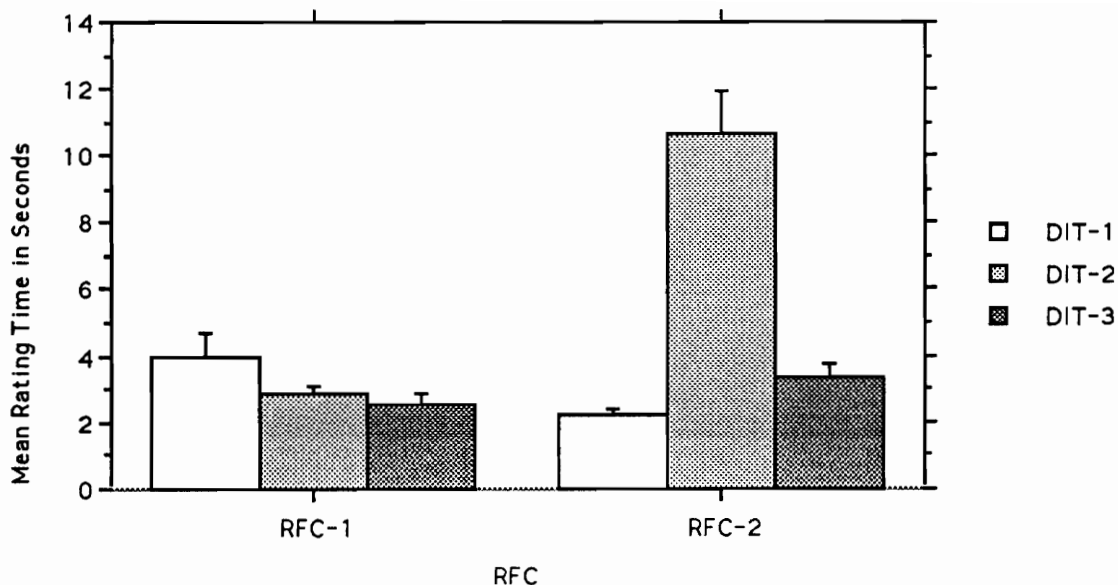


Figure 17. Mean rating times for the DIT X RFC interaction with standard error bars.

Again, a Newman-Keuls analysis was conducted to determine which levels of the DIT X RFC interaction are significantly different. The only significant differences found are all related to the DIT-2/RFC-2 condition. The DIT-2/RFC-2 condition differed significantly from all other DIT X RFC conditions by being much slower (i.e., the subjects required a lot more time to make their reusability decision for DIT-2/RFC-2 classes).

Thus, it appears that the RFC and DIT main effects are driven entirely by the DIT-2/RFC-2 cell. One can conclude from this finding

that the main effects are probably of little importance due to the nature of this interaction.

Task 2 Errors. The errors that occurred during class location (i.e., the wrong class was selected by the subject) were also recorded and analyzed with an ANOVA (see Table 29). The only significant main effect is RFC ($F(1, 42) = 17.02, p = 0.0002$).

TABLE 29

ANOVA Summary Table for Task 2 Errors

Source	df	Mean Square	F	p	P_{G-G}	ϵ_{G-G}
Programmer (P)	1	3.3368	.9600	.3328		
Representation (R)	2	5.9479	1.7113	.1930		
P * R	2	.6701	.1928	.8254		
Subject(Group) (S/G)	42	3.4757				
DIT	2	1.6354	2.1027	.1285		
DIT * P	2	.4618	.5938	.5546		
DIT * R	4	1.8333	2.3571	.0601		
DIT * P * R	4	1.2014	1.5446	.1968		
DIT * S/G	84	.7778				
RFC	1	20.5868	17.0164	.0002		
RFC * P	1	1.8368	1.5182	.2247		
RFC * R	2	1.4618	1.2083	.3089		
RFC * P * R	2	.1701	.1406	.8692		
RFC * S/G	42	1.2098				
DIT * RFC	2	80.8368	85.9531	.0001	.8828	
DIT * RFC * P	2	4.6493	4.9436	.0124	.8828	
DIT * RFC * R	4	1.4618	1.5543	.1941		
DIT * RFC * P * R	4	.7951	.8455	.5003		
DIT * RFC * S/G	84	.9405				
Total	203					

The error means for RFC are contained in Table 30. RFC-2 classes resulted in significantly more locating errors than RFC-1 classes in Task 2.

TABLE 30

Mean Error Rates for the RFC Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
RFC-1	144	2.2639	1.4435	.1203
RFC-2	144	2.7986	1.3514	.1126

The only significant two-way interaction for Task 2 errors was the DIT X RFC interaction ($F(2, 84) = 85.95, p_{G-G} = 0.0001$). The means for the DIT X RFC interaction are contained in Table 31 and illustrated in Figure 18.

TABLE 31

Mean Error Rates for the DIT X RFC Interaction Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1	48	2.4375	1.1833	.1708
DIT-1, RFC-2	48	2.8750	1.0237	.1478
DIT-2, RFC-1	48	1.3333	1.4489	.2091
DIT-2, RFC-2	48	3.7500	.8121	.1172
DIT-3, RFC-1	48	3.0208	1.1576	.1671
DIT-3, RFC-2	48	1.7708	1.3565	.1958

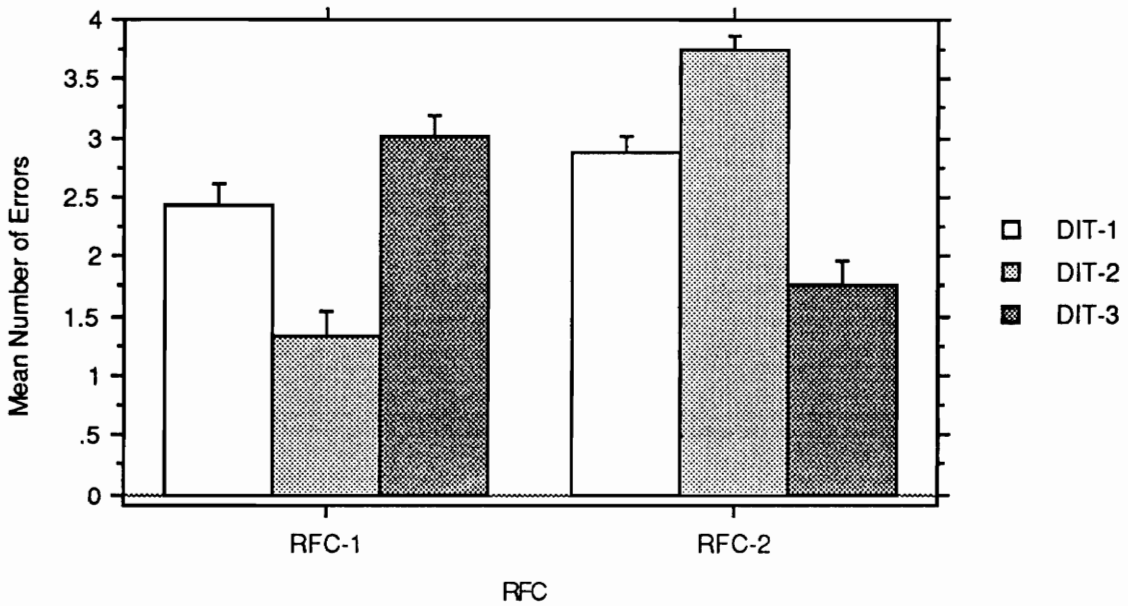


Figure 18. Mean errors for the DIT X RFC interaction with standard error bars.

The Newman-Keuls analysis is summarized in Table 32. All of the paired comparisons are significantly different except for the DIT-3/RFC-1 versus DIT-1/RFC-2 or DIT-2/RFC-2, and the DIT-1/RFC-2 versus DIT-1/RFC-1 comparisons.

DIT-2/RFC-1 resulted in the fewest errors while DIT-2/RFC-2 resulted in the most errors. DIT-3/RFC-1, however, exhibited the exact opposite trend and had more errors than DIT-3/RFC-2.

TABLE 32

Newman-Keuls Post-Hoc Analysis for DIT X RFC Interaction for Error Rates, Task 2

	DIT-2 RFC-1	DIT-3 RFC-2	DIT-1 RFC-1	DIT-1 RFC-2	DIT-3 RFC-1	DIT-2 RFC-2
DIT-2 RFC-1	- - - -	Sig.	Sig.	Sig.	Sig.	Sig.
DIT-3 RFC-2		- - - -	Sig.	Sig.	Sig.	Sig.
DIT-1 RFC-1			- - - -	not sig.	Sig.	Sig.
DIT-1 RFC-2				- - - -	not sig.	Sig.
DIT-3 RFC-1					- - - -	not sig.

The final significant interaction is a three-way interaction for DIT X RFC X Programmer ($F(2, 84) = 4.94, p_{G-G} = 0.0001$). The means are contained in Table 33 and plotted in Figure 19.

TABLE 33

Mean Error Rates for the DIT X RFC X Programmer Interaction Conditions, Task 2

	Count	Mean	Std. Dev.	Std. Error
DIT-1, RFC-1, Stu	24	2.4583	1.0624	.2169
DIT-1, RFC-1, Pro	24	2.4167	1.3160	.2686
DIT-1, RFC-2, Stu	24	2.7917	.9315	.1901
DIT-1, RFC-2, Pro	24	2.9583	1.1221	.2290
DIT-2, RFC-1, Stu	24	1.0417	1.2676	.2588
DIT-2, RFC-1, Pro	24	1.6250	1.5829	.3231
DIT-2, RFC-2, Stu	24	3.7083	.8065	.1646
DIT-2, RFC-2, Pro	24	3.7917	.8330	.1700
DIT-3, RFC-1, Stu	24	3.2083	1.0624	.2169
DIT-3, RFC-1, Pro	24	2.8333	1.2394	.2530
DIT-3, RFC-2, Stu	24	1.3333	1.0901	.2225
DIT-3, RFC-2, Pro	24	2.2083	1.4738	.3008

Stu: Student
Pro: Professional

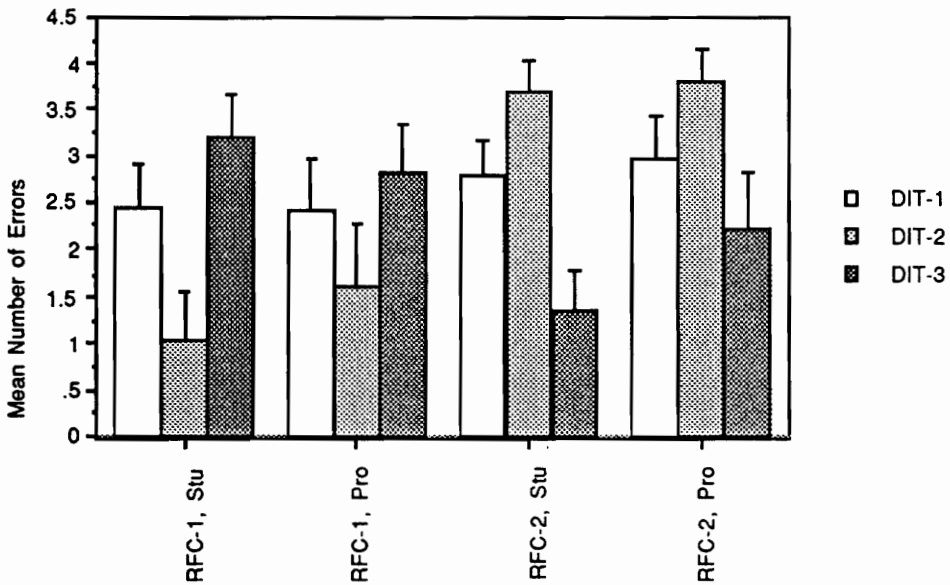


Figure 19. Mean errors for the DIT X RFC X Programmer interaction with standard error bars.

A simple-effects test was also conducted to isolate where significant two-way interactions existed. Unfortunately, all of the two-way interactions proved significant. Thus, no real guidance was supplied to aid in the Newman-Keuls analysis interpretation.

The Newman-Keuls analysis is summarized in Table 34. Some of the interesting comparison results are that student and professional programmers did not exhibit a significant difference for any of the following comparisons: DIT-2/RFC-1, DIT-3/RFC-2, DIT-1/RFC-1, DIT-1/RFC-2, DIT-2/RFC-2, and DIT-3/RFC-1.

TABLE 34

Newman-Keuls Post-Hoc Analysis for DIT X RFC X Programmer
Interaction for Error Rates, Task 2

	SD2 R1	SD3 R2	PD2 R1	PD3 R2	PD1 R1	SD1 R1	SD1 R2	PD3 R1	PD1 R2	SD3 R1	SD2 R2	PD2 R2
SD2 R1	- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
SD3 R2		- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.	Sig.
PD2 R1			- - -	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.	Sig.	Sig.
PD3 R2				- - -	not sig.	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.	Sig.
PD1 R1					- - -	not sig.	not sig.	not sig.	not sig.	not sig.	Sig.	Sig.
SD1 R1						- - -	not sig.	not sig.	not sig.	not sig.	not sig.	Sig.
SD1 R2							- - -	not sig.	not sig.	not sig.	not sig.	not sig.
PD3 R1								- - -	not sig.	not sig.	not sig.	not sig.
PD1 R2									- - -	not sig.	not sig.	not sig.
SD3 R1										- - -	not sig.	not sig.
SD2 R2											- - -	not sig.

DISCUSSION

Task 1

Task 1 RT. The results from the Task 1 analysis indicate that the representation hypothesis is incorrect. In fact, the exact opposite result was obtained. The modal block clustered representation was worse than the class/subclass tree and the hierarchical cluster analysis tree because the time to locate classes was greater than for either of the other two representations. The class/subclass tree and the hierarchical cluster analysis tree were not found to be significantly different based on the response time differences.

It appears that, at least in this simple search situation, neither of the two reorganization methods aid user performance as measured by response time. Thus, neither of the two reorganization methods appears useful for a simple search situation.

The Task 1 analysis results are somewhat perplexing given the positive performance that both the modal block clustering method and the hierarchical cluster analysis method have shown in the past (Shurtleff et al., 1988; Tullis, 1985). One explanation for this discrepancy may be that the past applications of both the modal block clustering method and the hierarchical cluster analysis method

have been in organizing haphazardly arranged menus and help systems.

The current research examined software that was already organized in a manner based on inheritance. Any reorganization for a generic search task may just be distorting the existing useful information structure.

Also, the components of the systems studied in the past were not as interrelated as are the classes and subclasses of an object-oriented system. The menu items and help listings were discrete unrelated units that were meant to function together for a single purpose (e.g., feature navigation).

For example, Tullis (1985) was actually imposing a logical tree structure on a large group of uncategorized features by using hierarchical cluster analysis. Thus, a generic search task would be quicker because hierarchical cluster analysis has imposed some logical structure on a haphazard collection of features. For the user interface library in this study, however, there already exists a logical structure and therefore no improved locating time was found.

It is possible that different sorting of data by more expert sorters may have resulted in a better showing for the hierarchical cluster analysis method. This does not seem likely, however. Therefore, the type of performance results obtained suggests that the

class/subclass tree allowed better search performance by the subjects in this study.

An attribute-based reorganization (such as modal block clustering) applied to an object-oriented library may not be effective for generic search tasks because the classes in a class/subclass tree already share a number of attributes if their relationship is based on inheritance. The attribute reorganization may reshuffle the class arrangements based on less important attributes, resulting in poor search times for a generic search task.

If, however, a user were searching based on a particular attribute, then the attribute-based reorganization may be useful. The use of an attribute-based reorganization could be linked to Prieto-Diaz's work on faceted classification, which is also attribute-oriented (Prieto-Diaz and Freeman, 1987). Thus, it is possible that a different search task (based on attributes) could have resulted in different findings. One can conclude from this discussion that more research is needed to compare specific representation structures (based on mental model techniques) with specific types of search tasks.

One way to address the inappropriate reshuffling of classes issue may be to block classes and then organize them into trees or

blocks within blocks. This seems overly involved for smaller systems, however, but it might be appropriate for large systems.

Another approach would be to study what level of attributes to use in the blocking. A better blend of abstract and concrete attributes may result in better subject performance.

The significant main effects for DIT and RFC agree with what one might expect from Lei's (1991, 1993) research. He found that increased DIT and RFC increased the difficulty of maintenance activity for programmers.

The current data indicate that search times increase as classes are located farther down into the class/subclass tree. Not surprisingly, search times are greatest for classes that do not exist. The long search times, however, may be due to the experimental setting. The subjects may have felt compelled to conduct a more careful search than they normally would because of the artificial nature of a study.

As systems such as the TI Explorer have demonstrated, programmers are often quick to write new code (see also Woodfield, Embley, and Scott (1987), who found that programmers are poor estimators of when to reuse code versus when to write new code). Programmers, in all likelihood, would terminate their search more quickly in a "real-world" setting.

Search times also increase for classes that are more highly connected (where connectivity includes inheritance relations and method calls) to other classes (i.e., RFC-2 classes). This result may be due to more visual clutter, but that point would need further study to determine its validity.

The DIT X Representation interaction clearly shows for the classes that do not exist (i.e., DIT-4) that search times are longest for modal block clustering, followed by hierarchical cluster analysis and class/subclass trees (see Figure 8). Thus, while searches for classes that did not exist took longer than all other searches regardless of representation, the times to locate DIT-4 classes are even longer for the reorganized class libraries than for the original class/subclass tree.

The RFC X Programmer interaction is also not only significant but very interesting. There were no significant differences found between students and professionals when comparing RFC-1 classes and again when comparing RFC-2 classes (Table 8 and Figure 9). One would expect professionals with their well developed programming skills and schemas (Mayer, 1981) to more quickly locate simple items.

Also, professional programmers had no significant difference between RFC-1 and RFC-2 classes but the student programmers did

have a significant difference between RFC-1 and RFC-2 classes. This finding fits with Mayer's (1981) line of logic.

The significant difference between the RFC-1 and RFC-2 search times for the student programmers may also support the Boehm-Davis, et al. (1986) and Holt, et al. (1987) findings (that program structure has a larger effect on students than on professionals). More research is needed to study why RFC-2 classes increased student search times so much. It is possible that RFC-2 classes had more links to other classes, thereby increasing visual clutter.

Finally, the DIT X RFC interaction was significant. DIT-4 (i.e., class does not exist) was the worst level of the DIT factor as measured by search performance. The DIT-4/RFC-2 and DIT-4/RFC-1 conditions did not significantly differ, which is what one would expect because there can be no higher or lower levels of connectivity for a class that does not exist.

Also of note is that the DIT-2/RFC-2 condition had the longest search time. More specifically, DIT-2/RFC-2 also had a larger search time than DIT-3/RFC-2. This indicates that RFC-2 classes in the middle of the tree (i.e., DIT-2) are more difficult to locate than RFC-2 classes at the end of the tree (DIT-3). Again, this could be due, to some extent, to visual clutter.

Finally, the results show that RFC-2 classes take longer to locate than do RFC-1 classes for DIT-1 and DIT-2. This is not the case for DIT-3, however. A possible explanation for this finding is that a programmer has already visually navigated through the library representation structure and it does not matter whether he or she finishes on an RFC-1 or RFC-2 class. If he or she has tracked through the representation structure correctly, then he or she has arrived at the correct class. It is also possible that the search times are large enough, due to getting lost in the representation structure, that it does not matter whether the final class in DIT-3 is an RFC-1 or RFC-2 class because the search has become an exhaustive search.

It is worth noting that the RFC range, while large, is only broken into two portions. A useful study would be to further break down the RFC range into smaller segments. This would be especially useful for the RFC-2 portion of the range. Such a study could more accurately identify where in the RFC-2 range that the higher connectivity begins to interfere with class locating performance. The identification of a "break point" would generate a useful "rule-of-thumb" for software design.

Task 1 Errors. The findings for Representation, when examined in the light of error rates, are the opposite of the RT data (i.e., class/subclass > modal block clusters). This may occur because

longer search times may lead to more careful searching and thus reduced errors.

Thus, the error rate results for modal block clustering, while attractive on the surface, may simply be related to the inflated search time (relative to class/subclass). The subjects invested more time to find classes so they made sure they had the correct class.

This situation could easily change in the "real world" where the programmer knows that no one is measuring his search times or error rates. Usually, there is not an absolutely correct answer to compare against. Also, the reward structure is not in place to reward programmers to reuse code. Accolades are collected by creating new code and not by reusing someone else's code. Therefore, error rate is probably not as useful as it would appear at first glance.

The DIT findings for error rates are compatible with Lei's (1991, 1993) software maintenance findings (i.e., DIT-3 > DIT-2 > DIT-1). An interesting finding is that DIT-4 (i.e., classes that do not exist) had the lowest error rates. Again, it may be the case that longer search times lead to more careful searching and thus lower error rates.

The RFC findings are also in agreement with Lei's (1991, 1993) results. The error rates for RFC-2 classes are higher than for RFC-1 classes.

The RFC X Representation findings do not agree with the speculation that longer search times lead to reduced errors. The RFC-2 classes required greater search times but RFC-2/class subclass has significantly more errors than all of the RFC-1 classes and RFC-1/modal block cluster has less errors than RFC-2/(hierarchical cluster analysis and modal block cluster). These findings do not agree with the time error speculation.

Finally, the DIT X RFC interaction results are not supportive of the tradeoff between errors and search time. The DIT-2/RFC-2 condition, which had the highest RT (for the DIT X RFC interaction), also had the highest error rate. In fact, DIT-3 for RFC-1 and RFC-2 had the second and third highest error rates (again, they also had large RTs). The tradeoff between errors and time may break down when considered in the middle of a representation (i.e., DIT-2) and at the end or bottom of a representation (i.e., DIT-3).

Also of interest is that the DIT-4/RFC-1 and DIT-4/RFC-2 error rates are not significantly different. This is appropriate because there is no difference in connectivity for classes that do not exist. This finding is in line with the RT data, which also found no difference between the pair.

Task 2

Task 2 RT. As was expected, overall search times were slower in Task 2 than in Task 1. For example, compare the cell means of reaction time for the DIT X RFC interaction for both Task 1 and Task 2.

The only main effect that is significant in the search time component of Task 2 is Programmer. This is an interesting finding that was not expected. Student programmers were found to complete the class location searches faster than professional programmers.

There are several possible explanations for this finding. One is that professional programmers are "out of practice" in interpreting questions such as those in Task 2. The second task used a word problem format that closely resembles a test. It may be possible that Task 2 simply illustrates that the professional programmers had "rusty" test-taking skills, which would lead to slower response times.

Another possibility is that the professional programmers may be trying to be more careful. Their experiences with complex systems may lead them to be more methodical and careful in their target search and selection.

Motivation does not seem to be an issue. Both groups, students and professionals, seemed equally motivated. It is possible that the professionals had an even higher motivation level than the students because they were "true" volunteers (i.e., no reward other than interest in the topic, whereas the students were awarded bonus points in their class).

The only two-way interaction that was significant was the DIT X RFC interaction. The results for this interaction were somewhat confusing. The DIT-2/RFC-1 condition had the shortest RT of all. On the other hand, DIT-2/RFC-2 had the longest RT of the group (except for DIT-1/RFC-1). While the second result makes sense (higher connectivity in the middle of the representation is more difficult to find due to clutter), the first result does not make sense.

One possibility to explain this result is the increased semantic or meaning component of the search in Task 2. The search activity was no longer a simple pattern match but rather it required an active cognitive component in determining what was the correct class to select.

The increased connectivity could significantly slow down the decision process in selecting a class. Reduced connections (or neighboring classes) may speed the locating and selecting process.

Task 2 Rating. Ratings was included in the analysis as a "check." The only effect that is significant is RFC. RFC-2 was rated less reusable than RFC-1.

Programmers may feel that classes with higher connectivity are to be avoided (i.e., potential errors from links to other software) or that they may be too generic to be very reusable. More research needs to be conducted on programmer rating of class reusability.

Potentially, the rating dependent variable is the most important measurement in this study. Although it is not a performance based measure, it is critical from the programmer's perspective. The use of new methods and tools often hinges on user perceptions rather than small performance differences in response time. Thus, a new way may be faster or involve fewer errors, but if users cannot effectively judge the reusability of the classes then the reorganization will not be useful or wanted (and this would be reflected in the ratings and not in the response times or error rates).

This study did not find an effect for representation, but this could change if a different task is used. Clearly, this is an important check because even if a programmer can find a class, it may not be readily apparent to him that he or she can use a class. The limited amount of research that has been done on programmers' accessing reusability has not been encouraging (e.g., Woodfield et al., 1987).

Task 2 Rating RT. The analysis of the time to make the reusability rating decision was added based on the idea that rating response time may indicate how sure the subjects were about their ratings. The assumption is that quicker ratings are an indication of higher conviction (i.e., more sure) while slower ratings indicate a rating that carries less conviction (i.e., less sure).

Subjects could have been directly asked about their confidence level but this act would have extended the study beyond a reasonable time slot for the volunteer subjects being used. The Rating RT was unobtrusively captured and did not add to the overall time of the study. Future research should consider the degree of relationship for these two measures in the area of software reusability assessment.

The degree of confidence that a subject has in his rating of a class may affect how much of the class would actually be reused. If a user believes that a class will not be very reusable, then he is likely to not reuse much of that class (regardless of whether the class could actually be reused or not).

The main effects of DIT and RFC were both found to be significant. The DIT-2 level had the slowest time to rate of the three DIT levels. It is also interesting to note that the standard error is also approximately three times larger than for the other two DIT

levels. One possible interpretation for the slower time to rate reusability is that the visual clutter of surrounding classes in the tree, even though the other classes may not be connected, caused the subjects to more thoroughly evaluate their reusability decision (i.e., less sure about rating for DIT-2)

The RFC main effect showed that the RFC-1 level is significantly faster than the RFC-2 level. The greater connectivity of RFC-2 may make the subjects evaluate more carefully the possible reusability of the class (i.e., less sure about rating).

The DIT X RFC interaction is also statistically significant. Again, the DIT-2/RFC-2 level was a problem. The DIT-2/RFC-2 condition represents a highly connected class in the middle of the class/subclass tree. The subjects may have perceived these types of classes as an abstract class that one adds to or modifies to get the class to accomplish something useful. Often this description is fairly appropriate.

The classes on the ends of the tree tend to be largely usable. The classes at the start of the tree are templates that one would build on to create a new class. Most of the contents of a DIT-1 class would be reused because classes at the start of the tree tend to be fairly low level.

Classes at the end of the tree (e.g., a DIT-3), however, tend to function as plug-and-play types of classes. A good example would be a specific type of dialog box that is sent a text string to display to the user.

Task 2 Errors. The errors for Task 2 have a significant main effect for RFC. There were more errors associated with RFC-2 classes than with RFC-1 classes. A likely explanation is the higher connectivity of RFC-2 classes. This result also relates to the longer times required to make a reusability rating in Task 2 for the RFC-2 classes. Thus, the subjects not only required more time to make a reusability decision (i.e., less sure about the rating), they also made more errors in selecting the correct class.

The DIT X RFC interaction is also significant. An interesting finding is that DIT-2/RFC-1 < other conditions < DIT-2/RFC-2. This is the same pattern that occurred in the Task 2 RT. For the DIT X RFC interaction for Task 2 it appears that the two end cases for the range of conditions are the same (i.e., low error/low time and high error/high time). Thus, quicker locating times are associated with lower error rates. This is a different pattern than that found in Task 1. Perhaps this is a result of a cognitive component being added to the search process.

The last interaction, DIT X RFC X Programmer, was also statistically significant. This is a difficult result to interpret. One interesting item is that student programmers made more errors than professional programmers in the DIT-2/RFC-1 and DIT-3/RFC-2 conditions. This difference might be related to the finding that student programmers are faster than professional programmers. It is possible that students were faster by trading off some accuracy for speed.

Thus, to say that students performed better should be tempered. At worst, however, students performed no worse than professional programmers in this study. Thus, students appear to be acceptable substitutes for professional programmers in this type of class search task.

One approach to using the findings of this study would be to treat the student results as boundary definitions for professional programmer performance. The student class locating speed could function as the lower boundary for professional programmer class-locating time. At the same time, the student error rates could be used as the upper boundary for professional programmer error rate performance in class-locating tasks.

A number of the other conditions (for DIT X RFC X Programmer) showed no significant differences between student and

professional programmers: DIT-1/RFC-1, DIT-1/RFC-2, DIT-3/RFC-1, and DIT-2/RFC-2. This suggests more questions than it answers. More research needs to be conducted to study the types of search tasks, programmer type, and software metrics such as RFC and DIT.

Finally, it may be the case that to reorganize the entire class tree is a misguided effort. The results, when viewed as a whole (RT and error for both tasks), indicate that the middle levels of a class hierarchy (DIT-2 in Tasks 1 and 2) are more difficult to locate. User locating performance, however, seems acceptable around the "edges" of the class/subclass tree. The real problem may be to reorganize only the middle of the tree. The reorganization would need to not only preserve the hierarchy information but also augment it so that the user can more easily locate items. Clearly, this will not be an easy task.

The start and end of the class/subclass tree might be treated as entry points into a search space that is reorganized in a more efficient manner. More research would be needed to further develop this approach.

Another approach might be to leave the class/subclass tree alone, and concentrate on an attribute-locating tool. The tool might operate in an "attribute space" that could be organized by a modal

block cluster but the tool would put the user back into the class/subclass representation when it had found a potential target.

Thus, it may be the case that the class/subclass tree representation is also a proper mental model for working with classes (and inheritance). However, there is a need for a better set of tools to work with the class/subclass tree.

CONCLUSIONS

The first conclusion one can draw from the data is that the hierarchical cluster analysis method and especially the modal block clustering method did not work well when considered from a search time criterion.

If one also considers error rates, then the situation still does not change the conclusions. The hierarchical cluster analysis had an error rate and search times that were no worse than the class/subclass tree.

Unfortunately, however, conducting a hierarchical cluster analysis requires an enormous amount of work and relies on the presence of experts. For a new system, the experts may not exist. Also, even if the experts exist, a user interface designer may not have the time and resources to conduct a hierarchical cluster analysis. Thus, hierarchical cluster analysis is not an attractive option given the performance (i.e., equal search times and error rates).

The modal block clustering approach fared even worse. It actually degraded user performance by increasing RT. The interaction of the modal block clustering method with classes from an object-oriented system seems highly negative for a generic search

task. The results might be different for an attribute-based search task but this requires more study.

The error rates for modal block clustering were lower than were the error rates for class/subclass tree in Task 1. This, however, does not justify conducting a modal block cluster reorganization of a class/subclass tree. If other confirming evidence also existed (e.g., the same significant error rate difference in Task 2), then the error rate finding in Task 1 might be of more interest.

A serious problem for modal block clustering seems to be that class trees based on inheritance share features from parent to child. A modal block cluster muddles the class relationships by destroying the hierarchy and replacing it with large blocks of very similar classes but not showing how the classes in the blocks are related.

The second conclusion that can be drawn is that student programmers performed no worse than professional programmers on the simple search task (i.e., Task 1). The student programmers performed even better than the professional programmers in the complex search task (i.e., Task 2).

Task 2 showed that student programmers actually located classes faster than professionals in a complex search task. Thus, even with the caveats noted in the discussion (rusty test-taking skills and speed-error trade-offs), student programmers appear to be

acceptable substitutes for professional programmers in this type of class search task.

The third conclusion that can be drawn from the current research is that the classes in the middle levels of a class hierarchy (i.e., DIT-2 in Task 1 and Task 2) are more difficult to locate.

This presents a serious problem for users of large systems where most of the "volume" of the system is in the middle. This finding, however, may indicate where the "reorganization" research should go from here.

The final conclusion that can be drawn is that higher RFC levels (i.e., connectivity) sometimes reduce search times for classes on the front end of the class/subclass tree (i.e., probably due to the reading bias of left to right) but always increase search times for classes in the middle of the tree. Higher RFC levels also reduce the user reusability ratings and the subject's confidence in his or her assessment rating of the classes reusability.

The reduced reusability rating may be valid, however. The higher RFC may indicate that a class is an abstract class that needs to be heavily augmented or changed to accomplish something. This topic needs further study.

REFERENCES

- Boehm-Davis, D., Holt, R., Schultz, A., and Stanley, P. (1986). The role of program structure in software maintenance. Technical Report TR-86-GMU-P01. Fairfax, VA: George Mason University.
- Caramazza, A., Hersh, H., and Torgerson, W. (1976). Subjective structures and operations in semantic memory, *Journal of Verbal Learning and Verbal Behavior*, **15**, 103-117.
- Card, S., Moran, T., and Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, New Jersey: Lawrence Erlbaum.
- Carroll, J., and Olson, J. (1988). Mental models in human-computer interaction. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 45-65). New York: North-Holland.
- Chidamber, S., and Kemerer, C. (1991). Towards a metrics suite for object oriented design. In *Proceedings: OOPSLA '91*. New York: ACM.
- Cooke, N., Durso, F., and Schvaneveldt, R. (1985). Measures of memory organization and recall. (Tech. Report MCCS-85-11). Las Cruces, New Mexico: Computing Research Laboratory.
- Curtis, B. (1989). Cognitive issues in reusing software artifacts. In T. Biggerstaff and A. Perlis (Eds.), *Software reusability; Volume II: Applications and experience* (pp. 269-287). Reading, Massachusetts: ACM Press and Addison-Wesley.
- Davis, A., Bersoff, E., and Comer, E. (1988). A strategy for comparing alternative software development life cycle models, *IEEE Transactions on Software Engineering*, October, 1453-1461.

- Gentner, D. and Stevens, A. (Eds.) (1983). *Mental models*. Hillsdale, New Jersey: Lawrence Erlbaum.
- Grabow, P., and Noble, W. (1988). Development methodologies, In R. Thayer and M. Dorfman (Eds.), *System and software requirements engineering* (pp. 539-542). Washington: IEEE Computer Society Press.
- Greenhouse, S., and Geiser, S. (1959). On methods in the analysis of profile data. *Psychometrika*, **24**, 95-112.
- Helander, M. (Ed.) (1988). *Handbook of human-computer interaction*. New York: North-Holland.
- Holt, R., Boehm-Davis, D., and Schultz, A. (1987). Mental representations of programs for student and professional programmers. In (Eds.) G. Olson, S. Sheppard, and E. Soloway, *Empirical studies of programmers*. (pp. 33-46). Ablex: Hillsdale, NJ.
- Jorgensen, A. (1987). The trouble with UNIX: Initial learning and experts' strategies. In (Eds.) H. Bullinger, and B. Shackel, *Human-computer interaction-INTERACT '87*, (pp. 847-854). Elsevier Science Publishers: New York.
- Lei, W. (1991). *Applying software complexity metrics in the object-oriented software development life cycle*. Unpublished doctoral dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.
- Lei, W., and Henry, S. (1993). Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, **23**, 111-122.
- Mayer, R. (1981). The psychology of how novices learn computer programming. *ACM Computing Surveys*, **13**, 121-141.

- McClure, C. (1992). *The three r's of software automation: Re-engineering, repository, reusability*. Englewood Cliffs, New Jersey: Prentice-Hall.
- McDonald, J., Stone, J., Liebelt, L., and Karat, J. (1982). Evaluating a method for structuring the user-system interface. In *Proceedings of the 26th annual meeting of the Human Factors Society*. Santa Monica, CA: Human Factors Society.
- McDonald, J., Dearholt, D., Paap, K., and Schvaneveldt, R. (1986). A formal interface design methodology based on user knowledge. In *Proceedings of the human factors in computing systems conference*. New York: Association for Computing Machinery.
- Norman, D. (1983). Some observations on mental models. In D. Gentner and A. Stevens (Eds.), *Mental models*, (pp.7-14). Hillsdale, New Jersey: Lawrence Erlbaum.
- Norman, D. (1986). Cognitive engineering. In D. Norman and S. Draper (Eds.), *User centered system design*, (pp. 31-62). Hillsdale, New Jersey: Lawrence Erlbaum.
- Prieto-Diaz, R. (1987). Domain analysis: An introduction. *ACM SIGSOFT - Software Engineering Notes*, 15 (2), 47-54.
- Prieto-Diaz, R., and Freeman, P. (1987). Classifying software reusability. *IEEE Software*, 4, 6-16.
- Rosch, E., and Mervis, C. (1975). Family resemblance studies in the internal structure of categories. *Cognitive Psychology*, 7, 573-605.
- Roske-Hofstrand, R., and Paap, K. (1986). Cognitive networks as a guide to menu organization: An application in the automated cockpit. *Ergonomics*, 29, 1301-1312.

- Shurtleff, M., Jenkins, J., and Sams, M. (1988). Deriving menu structures through model block clustering: A promising alternative to hierarchical techniques. In *Proceedings of the 32nd annual meeting of the Human Factors Society*. Santa Monica, CA: Human Factors Society.
- Smith, E., and Medin, D. (1981). *Categories and concepts*. Cambridge, Massachusetts: Harvard University Press.
- Tracz, W. (1992). Domain analysis working group report - first international workshop on software reusability. *ACM SIGSOFT - Software Engineering Notes*, 17 (3), 27-34.
- Tullis, T. (1985). Designing a menu-based interface to an operating system. In *Proceedings of the human factors in computing systems conference*. New York: Association for Computing Machinery.
- Waddington, R., and Henry, R. (1990). Expert programmers re-establish intentions when debugging another programmer's program. In (Eds.). D. Diaper, D. Gilmore, G. Cockton, B. Shackel, *Human-computer interaction-INTERACT '90*, (pp. 965-970). New York: Elsevier Science Publishers.
- Wickens, C. (1984). *Engineering psychology and human performance*. Columbus, OH: Merrill.
- Woodfield, S., Embley, D., and Scott, D. (1987). Can programmers reuse software? *IEEE Software*, 7, 52-59.
- Young, R. (1983). Surrogates and mappings: Two kinds of conceptual models for interactive devices. In D. Gentner and A. Stevens (Eds.), *Mental models*, (pp. 35-52). Hillsdale, New Jersey: Lawrence Erlbaum.

Appendix A

This appendix contains all the class names that were searched for by the subject in Task 1. Also included are the DIT metric and RFC metric values for the classes.

Command Name	DIT	RFC
object	0	1 2
data	1	7
action	1	1 4
context	1	2 1
fw_phase_action	1	2 2
gio_action	1	2 2
tools_action	1	2 7
role_select_window	1	3 5
screen	1	4 6
report_view_window	1	4 8
graphic	1	6 7
proj_definition_action	1	9 0
cancel	2	2
done	2	2
quit	2	2
rectangle	2	1 1
line	2	1 7
tools_window	2	1 7
role_select_data	2	3 5
string80_rendering	2	4 0
son_of_equation_data	2	4 7
framework_definition_data	2	8 6
fw_specific_data	2	1 1 0
son_of_abstract_data	2	1 5 6
popup_menu	3	9
dialog	3	1 2
indented_list	3	1 2
alert	3	2 0
selector	3	2 5
name_dialog	4	1 0

Command Name	DIT	RFC
menu_title	3	29
horizontal_view	3	30
vertical_view	3	30
report_field_rendering	3	37
report_field	3	109
property_dialog	4	37

Does Not Exist	DIT	RFC
persistent_float_data	4	1
annotation_data	4	1
personnel_action	4	1
about_dialog	4	1
date_dialog	4	1
check_box	4	1
decorated_view	4	2
scroll_bar	4	2
bar_graph	4	2
list_picker	4	2
spreadsheet	4	2
gio_icon	4	2
Practice Items		
dictionary		
ratio		
equation_data		
ques_data		
user_data		

Appendix B

This appendix contains all the class names that were searched for by the subject in Task 2. Also included are the specifications that the programmers use.

String80_Rendering

Your program will need to provide a method for visually rendering a `String80_Data` object on a display surface. Locate the class that would supply this capability.

Framework_definition_data

You will be using the `Fw_definition_window` class. Locate the data class that will support that window.

Role_select_data

You will be using the `Role_select_window` class. Locate the data class that will support that window.

Action (or Action_q)

You will be using a number of action classes that contain a single routine that is executed in response to a trigger message. Locate the action class that serves as a template (i.e., specifies the common behavior) for all the action subclasses.

Tools_window

Your program will need to create a window that allows the user to add, remove, and reconfigure tools. Locate a class that will create that window.

Alert

You will be using information displays that contain text and buttons and that notify the user when there is a potential problem. Locate the information display class that supports this function.

Cancel

You will be using cancel buttons in your dialog boxes and windows to stop different actions. Locate the class that supports this capability.

Line

Your program will be required to display a straight line on the display surface. Locate the class that would support this capability.

Context

Your program will need to store the context information needed by the graphical user interface. Locate the class that supports this capability.

Data

Your program will work with a number of different data classes. Locate the class that serves as a template (i.e., specifies the common behavior) for all the data objects.

Dialog

You will be using interactive information displays that contain text and buttons and that are mainly used to present the user with various choices. Locate the information display class that supports this function.

Report_view_window

Your program will need to provide a window so that the user may view reports. Locate the class that will create that window.

Graphic

Your program will work with a number of objects that are visible to the user on the visual display. Locate the class that is used as a template to specify the common behavior of all objects that are visible to the user.

Horizontal_View

Your program will be required to manage the visual presentation of information that is arranged end-to-end horizontally. Locate the class that would support this capability.

Menu_Title

Your program will be required to possess title bars with menu capabilities at the top of the screen. Locate the class that would support this capability.

Name_Dialog

Your program will be required to interactively name and rename items. Locate the dialog box that supports that capability.

Object

One class exists at the root of the object inheritance tree and defines the behavior inherited by all the objects. Locate this class.

Popup_Menu

Your program will need to create modal popup menus to interact with the user. Locate the class that supports this capability.

Quit

You will be using quit buttons in your dialog boxes and windows to terminate the application upon user request. Locate the class that supports this capability.

Rectangle

Your program will need to display rectangles on the display surface. Locate the class that supports this capability.

Selector

Your program will need to present a scrollable list of items from which a user can select specific items. Locate the class that supports this selection capability.

Vertical-View

Your program will be required to manage the visual presentation of information that is arranged top-to-bottom vertically. Locate the class that would support this capability.

Fw_Phase_Action

You will be using the Fw_phase_window class. Locate the action class that will support that window.

Project_definition_action

You will be using the Project_definition_window class. Locate the action class that will support that window.

Role_select_window

Your program will need to provide a window for role selection. Locate the class that will create that window.

Report_field

You will be using the Report_view_window class. Locate the class that will support the use of fields in that window.

Report_field_rendering

You will be using fields in the Report_view_window. Locate the class that will visually render the fields on the display surface.

Fw_Specific_Data

You will be using the Fw_specific_window class. Locate the data class that will support that window.

Tools_action

Your program will use the Tools_window class. Locate the action class that will support that window.

Son_of_equation_data

Your program will need to deal with data in the form of equations. Locate the class that will provide that capability.

Appendix C

This appendix contains all the class names in UIMS. Also included are the DIT metric and RFC metric values for each of the classes.

Command	DIT	RFC
action	1	14
alert	3	20
boolean_data	2	12
cancel	2	2
context	1	21
data	1	7
dialog	3	12
dictionary	2	10
done	2	2
enumeration_rendering	3	24
float_data	2	12
float_rendering	3	17
graphic	1	67
horizontal_view	3	30
indented_list	3	12
Integer_data	2	12
integer_rendering	3	9
line	2	17
list_data	2	46
menu_title	3	29
name_dialog	4	10
object	0	12
offset_ratio	3	12
popup_menu	3	9
popup_window	2	29
quit	2	2
ratio	2	15
rectangle	2	11
selector	3	25
screen	1	46
string80_data	2	11
string80_rendering	2	40
tf_boolean_rendering	3	17
tree	2	54
uims	0	32
vertical_view	3	30
view	2	101
window	1	57
yn_boolean_rendering	3	17

core_list_data	0	27
data_select_window	2	39
description_window	2	29
equation_data	2	96
equation_node_data	2	62
fw_abstract_data	2	150
fw_abstract_node_data	2	68
fw_abstract_window	2	59
framework_definition_data	2	86
fw_definition_window	2	40
fw_level_data	2	54
fw_level_select_window	2	38
fw_phase_action	1	22
fw_phase_data	2	68
fw_phase_window	2	32
fw_specific_data	2	110
fw_specific_window	2	41
gio_action	1	22
gio_definition_selector	2	22
gio_definition_window	2	42
gio_view_action	1	38
gio_view_window	2	55
list_looker	3	24
main_action	2	38
main_window	2	50
personnel_window	2	38
proj_abstract_data	2	149
proj_abstract_window	2	41
project_data	2	95
proj_definition_action	1	90
proj_definition_window	2	58
proj_level_data	2	54
proj_level_window	2	40
proj_role_data	2	34
proj_roles_window	2	42
proj_specific_data	2	118
proj_specific_window	2	40
property_dialog	4	37
ques_data	2	35
ques_persistent_string80_data	1	35
role_select_window	2	35

report_constraint_window	2	39
report_definition_action	1	43
report_definition_window	1	62
report_field	3	109
report_field_rendering	3	37
report_selector	2	30
report_view_window	1	48
son_of_abstract_data	2	156
son_of_abstract_node_data	2	125
son_of_abstract_window	2	41
son_of_definition_window	2	34
son_of_equation_window	2	47
son_of_equation_node_data	2	56
son_of_level_select_window	2	34
son_of_phase_data	2	88
son_of_phase_window	2	31
son_of_specific_data	2	121
son_of_specific_window	2	45
template_abstract_window	2	39
template_definition_window	2	34
template_level_select_window	2	35
template_phase_window	2	32
template_specific_window	2	39
tool_data	2	33
tool_element	2	30
tools_action	1	27
tools_window	2	17
user_data	2	78
user_role_data	2	62
user_roles_window	2	40

Appendix D

This appendix contains the situation statement for the subjects in Task 2.

Situation:

You have been hired as a new programmer to work on a Personal Information Management (PIM) tool for managers. The company you work for has a class library of previously written software. You need to locate the relevant software in the library based on the brief specifications given to you.

Be aware that:

- some of the class names use abbreviations (e.g., fw for framework)
- sometimes more than one class may be appropriate, pick the one you think is best
- a class exists for each of the specifications, but you can select 'does not exist' if you cannot find a class that seems appropriate.

Please work as quickly and accurately as possible.

VITA

Joseph Anthony Jenkins

EDUCATION

Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, January 1991 - December 1993.

Ph. D.: Industrial and Systems Engineering, Human Factors Engineering Option

Advisor: Dr. H. Snyder

Dissertation: Facilitating Software Reuse by Structuring the SPS User Interface Management System's Software Library

According to Programmer Mental Models

New Mexico State University, Las Cruces, NM 88003, August 1985 - December 1989.

M.A.: Engineering Psychology; minor: Computer Science

Houghton College, Houghton, NY 14744, August 1980 - May 1984.

B.S.: Psychology

WORK EXPERIENCE

BNR Inc., RTP, NC, Subscriber Services Market Planning, 1993 - present.

Member of scientific staff working on service definition and user interfaces for telephony products and services.

Virginia Polytechnic Institute and State University, Blacksburg, VA 24060, Displays and Controls Lab, 1991-1992.

Research assistant for H. Snyder: examined JPEG and MPEG. research conducted on a Texas Instruments internship

International Business Machines, Charlotte, NC, Human Factors, Systems Development Lab, 1990-1991.

Staff scientist working on user interfaces for CT2 product. Also performed icon design, testing and analysis.

Pacific Science and Engineering Group, Inc., San Diego, CA, 1989 - 1990.

Staff scientist working on user interface for Skill-Score Project. Also performed data analysis and project proposal writing tasks.

International Business Machines, San Jose, CA, Software Human Factors Center, Santa Teresa Lab, 1988 - 1989.

Staff scientist working on database and expert systems products.


Texas Instruments, Dallas, TX, User Systems Engineering Group, 1987 - 1988.

Member of technical staff working on user interface design for Electronic Data Interchange and Explorer function browser.

New Mexico State University, Las Cruces, NM 88003, 1985 - 1988.

Teaching assistant: experimental methods, consumer psychology, personality, perceptual processes, introduction to psychology.

Research assistant for K. Paap: examined language processes.



Joseph Anthony Jenkins