An Empirical Study of Representation Methods for Reusable Software Components

William B. Frakes and Thomas P. Pole

Abstract—An empirical study of methods for representing reusable software components is described. Thirty-five subjects searched for reusable components in a database of UNIX tools using four different representation methods: attribute-value, enumerated, faceted, and keyword. The study used Proteus, a reuse library system that supports multiple representation methods. Searching effectiveness was measured with recall, precision, and overlap. Search time for the four methods was also compared. Subjects rated the methods in terms of preference and helpfulness in understanding components. Some principles for constructing reuse libraries, based on the results of this study, are discussed.

Index Terms—Software reuse, experimentation, empirical methods, information storage and retrieval, reuse libraries, component indexing, keyword searching, faceted classification, enumerated classification, component understanding, database

I. INTRODUCTION

MANY organizations are currently building libraries of reusable assets. In doing this, they face important questions, such as what kind of implementation platform to use, how to handle security, and how to represent the components in the library so that they can be found and understood by potential users. To date, there has been little empirical study of these questions, leaving practitioners with little guidance.

This paper describes an empirical study of representation methods for reusable software components. A representation is a library classification method, knowledge representation method, or hypertext method, plus a user interface. We conflate the method with the interface because they are impossible to separate experimentally; a given representation method must have a presentation interface.

A database of UNIX tools was represented using four different methods: attribute-value, enumerated, faceted, and keyword. These methods were chosen because they are most representative of current practice. Subjects searched a set of 28 queries using the four methods, and provided lists of items that they felt matched the queries. The searching tool was Proteus [1], a reuse library tool that supports multiple representation methods.

Subjects’ answers were compared to those of UNIX experts, and this information was used to calculate recall and precision measures. The time taken to search each method was also evaluated. Subjects also filled out a survey that asked which methods they preferred, and about features of the different search methods. The study addressed the following.

1) Are any of the four methods more effective than others for helping people find reusable software components?
2) Are any of the four methods more efficient in terms of searching time for finding reusable software components?
3) Which methods do people prefer?
4) Are effectiveness, efficiency, and preference related to user programming and UNIX experience?
5) Are any of the methods more helpful than others for helping users understand reusable components?
6) Do the methods retrieve the same items?

This sort of information is needed by practitioners faced with the task of creating reusable libraries and by designers of reuse library systems.

A. Software Reuse

Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality [2]. Reuse has proven to be a complex area affected by many factors. Sources on the general issues of reuse are [3]–[6].

There are two basic technical approaches to reuse: parts-based and formal language-based. The parts-based approach assumes a human programmer integrating software parts into an application by hand. In the formal language-based approach, domain knowledge is encoded into an application generator or a programming language. Examples of the application generator approach are lex and yacc in the UNIX environment, and tools such as Genesis [7], a database generator. Examples of domain-specific programming languages are APL and SAS, which have mathematical and statistical domain knowledge encoded into the operators of the languages. The study reported here focuses on the parts-based approach.

To incorporate reusable components into systems, programmers must be able to find and understand them. If this process fails, then reuse cannot happen. Thus, how to index and represent these components so that they can be found and understood are two important issues in creating a reuse program. The role of representation in a reuse environment is summarized in Fig. 1. Reusable components are acquired...
Methods for representing reusable software, and systems to support those methods, have proliferated in recent years. These methods are drawn from three major areas: library and information science, artificial intelligence (AI), and hypertext [10]. To date, AI and hypertext systems have been used only experimentally. Fielded reuse library systems use library and information science methods. These methods are discussed in more detail below.

Despite all this work, there are doubts about the importance of representation methods for reuse. For example, [11] argues that some of the most successful reuse environments in Japan use very simple representation methods. He concludes from this that the representation method used is unimportant. Is it the case, then, that the method used to represent a reusable parts collection will make little difference to successful reuse?

One answer comes from a survey of engineers and managers in several aerospace companies [12]. Participants in the survey were asked to rate the importance of various reuse technologies. They rated library problems as being only moderately important to them, and as significantly less important than issues related to the acquisition of reusable assets. On the other hand, practitioners continue to list library problems as impediments to reuse [13].

The importance of library issues to a reuse program is dependent on several factors.

- Library issues will be less important in environments with low staff turnover, because the component information will be available from the people who work in the environment. This may be one reason why Japanese companies can successfully use simple descriptions of parts in their reuse programs. In Japanese companies, a typical engineer stays on the same project for 10 to 15 years. In the United States, the figure is about three years.
- Library issues may be less important in reuse environments based on generative methods, because these will require less human intervention, such as searching for and understanding components during the software construction process.
- The importance of representation techniques is minimal for small collections, because the searching problem is easier. Many organizations are just beginning to acquire reusable assets, and thus have small collections of at most a few hundred components. Large reuse collections are beginning to appear, however. Bell Northern Research maintains an on-line library of about 16 million lines of software [14]. The Asset library now being constructed will contain many thousands of components [15]. IBM has a distributed library containing more than 1200 components [16].

Representation methods are also important for reuse, because they help users understand components and application domains. Faceted classification, for example, is the base technology for the domain analysis method proposed by Prieto-Diaz [17]. The LaSSIE system [18] is another example of the use of representation methods as an aid to system understanding.

Given the growth in size of reuse libraries, the turnover rate of engineers at American companies, and the importance

<table>
<thead>
<tr>
<th>Level</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>Graphical Tree</td>
</tr>
<tr>
<td>Representation</td>
<td>Class Hierarchy</td>
</tr>
<tr>
<td>Implementation</td>
<td>Paper Manual</td>
</tr>
</tbody>
</table>

Fig. 1. Reuse library environment.
of representation methods for component and domain understanding, representation methods are, and will continue to be, an important issue in software reuse.

II. REPRESENTATION METHODS

As stated above, there are three classes of representation methods for reusable components: AI, hypertext, and library and information science methods. We studied the library and information science methods because they are the ones in use in industry today.

As Fig. 2[10] shows, library science methods break into two main categories: controlled vocabulary and uncontrolled vocabulary. A controlled vocabulary places limits on the terms that can be used to describe a classified object and/or on the syntax that can be used to combine those terms. A controlled vocabulary thesaurus, e.g., the Library of Congress Subject Headings used in most public library card catalogs, lists acceptable terms that can be used to describe things, and unacceptable terms that cannot be used.

Uncontrolled vocabularies do not place restrictions on the terms and syntax that can be used in a description. The methods that have been used for nearly all fielded reuse library systems are enumerated, faceted, and uncontrolled (free text) keyword.

In enumerated classification a subject area is broken into mutually exclusive, usually hierarchical, classes. The classification of library and information science methods above, for example, is an enumerated classification. Enumerated classification is very common. The Dewey decimal system for library classification is a well-known example. The following is an example of part of a scheme for classifying UNIX tools.

UNIX software
    directory operations
    create
    mkdir in
    destroy
    rmdir
    file operations
    create/modify
    edit
    ed, vi, view, emacs, ex, vedit

Here the domain of UNIX software is broken into two subclasses: directory operations and file operations. Directory operations is subdivided into the classes create and destroy. Create contains the tools mkdir and ln, and destroy the tool rmdir.

The fact that they are so highly structured makes enumerated classifications easy to understand and use. The well-defined hierarchy helps users understand the relationships among indexing terms, and provides a natural searching method of moving up and down the classification tree. The disadvantage of enumerated classification is that it requires that the indexing domain be completely analyzed and broken into exclusive hierarchical categories. This requires a lot of domain knowledge, and also limits the kinds of relationships that can be represented. The rigidity of the classification also makes the classification scheme difficult to change, which may be required as the domain evolves or becomes better understood.

In a faceted classification[19], a subject area is analyzed into basic terms that are organized as facets. These facets are then usually ordered from left to right, based on perceived importance. Objects are then classified by synthesizing the facet term pairs in the classification scheme. The development of facets is usually done by identifying important vocabulary in a domain and then grouping like terms together into facets.

A set of components, for example, might be defined with facets based on the object they operate on, the general operation they perform, and the specific activity they do, as shown in Table II.

A faceted classification scheme gives a classifier freedom to create complex relationships by combining facets and terms. It is also much easier to modify than is a hierarchical scheme, because one facet can be changed without affecting others in the classification scheme.
In attribute-value classification, a collection is described in terms of a set of attributes and their values. One might choose, for example, such attributes as action, functional area, language, and type of life-cycle object. Each attribute will have values. The action attribute, for example might have the values list, login, move, preprocess, search, sort, and so on. This method is similar to faceted searching in that facets are equivalent to attributes, and facet terms are equivalent to values. The following are the differences.

- One generally tries to describe a domain using seven or fewer facets. This limit is not typically used in defining attributes.
- There is typically no ordering of attributes and values, as one does with facets and terms.
- Faceted schemes usually provide some facility for handling synonyms not found in simple attribute-value searching.

In free text keyword indexing [9], [20], terms are automatically extracted from documentation, for example, the descriptive header on a code module. This method would fall in one of the two far-right leaves of the tree in Fig. 2. In an uncontrolled vocabulary, no restriction is placed on the terms that can be used to describe an item. Uncontrolled vocabulary terms can be drawn from any source, but are usually drawn from the indexed objects themselves. The following are some potential advantages of using an uncontrolled vocabulary.

- **Cost:** Since the index terms are often drawn directly from the text of the indexed objects, the indexing task can be highly automated. This is usually much cheaper than human indexing.
- **Specificity:** Since terms are unrestricted, indexing terms can be made as specific as possible. For example, a searcher looking for a linked-list algorithm may be instructed by a controlled vocabulary system to use the broader term list. An uncontrolled system would not dictate a decision of this kind.

### A. Related Studies

Though many papers have been published on reuse libraries and related issues (see [10] for a survey), little experimental evaluation of them has been done. Much of the information we do have comes from experiments with document collections [21]. The following are some of the major findings from this large body of work.

- No indexing method works really well. Typical recall and precision numbers are in the 40% to 60% range. Recall is the number of relevant items retrieved over the number of relevant items in the database. Precision is the number of relevant items retrieved over the number of all items retrieved. Recall and precision are the classic measures of the effectiveness of an information retrieval system.
It is difficult to significantly improve on simple retrieval methods, such as Boolean keyword, by using more sophisticated methods, such as vector space and extended Boolean [9].

Different representation methods will locate different items, though their recall and precision numbers tend to be similar [22].

People will avoid using systems that are difficult or inconvenient.

For reuse systems specifically, only a few studies have been done.

Prieto-Díaz [23] compared ordered versus unordered keywords in his dissertation and found that the ordered keywords performed better, but no statistical analysis was done. Maarek et al. [24] compared a free text phrase extraction method against free text keyword. They found that the phrases performed better, but provided no statistical analysis to determine if the improvement was significant. Reference [25] measured the recall and precision performance of keyword and natural language queries on three small parts databases and found that they achieved fairly high recall and precision, but, again, provided no statistical analysis.

III. EXPERIMENTAL DESIGN

In our study, each subject was given a set of seven queries for each representation method, and was asked to find all items in the database relevant to the query. Subjects spent approximately four hours on the experiment as follows:

- a half-hour introduction to the use of Proteus and explanation of the experimental task,
- three hours for searching using the four methods, and
- a half-hour to fill out the survey form.

These steps, and the data that resulted from them are summarized in Fig. 3.

A. Dependent Variables

The dependent variables for the study were recall, precision, search time, user preference, and helpfulness of the methods for understanding components. The null hypotheses were that
there is no difference between the representation methods for any of the dependent variables.

B. Independent Variables

The independent variable is representation method with four levels: attribute-value, enumerated, faceted, and keyword. Subject differences and levels of programming and UNIX experience were also examined.

C. Statistical Design

This experiment used a repeated measures factorial design [26]. The appeal of this design is that it increases the statistical power of the test for a given number of subjects. The survey used in this study also assumes that subjects will use all of the four methods in the study.

For a given subject, the four query sets (see Section V-D) were randomly assigned to the representation methods. This means that all query sets were searched under all methods, but all query sets were not searched under all methods by all subjects. The order in which subjects searched the representation methods was randomized to minimize learning or order biases.

IV. SUBJECTS

The subjects in this study were 35 employees of the Software Productivity Consortium, primarily software engineers and analysts. Their years of programming experience and UNIX experience are summarized in Fig. 4.

V. EXPERIMENTAL ENVIRONMENT

A. Proteus

Proteus [1] is a prototype software reuse library tool that supports multiple representation methods. Each of the methods in Proteus accesses the same low-level data structure that holds information about the parts. The middle level of Proteus consists of a set of functions that transform the lower-level data objects into the representations. Proteus supports four representations: keyword, faceted, enumerated, and attribute value.
All user input to Proteus is done via a three-button mouse. Left-clicking on an object selects it, and right-clicking brings up a help screen for the object. The middle mouse button is used in a few places for special search activities.

Fig. 5 shows the selection screen in the version of Proteus used in the experiments. The four methods are selected in order, and the sequence is then locked in for the course of the session. This was done to ensure that the subjects searched the methods in the randomly assigned order required for the experiment. In the example screen, attribute-value has already been searched, and is no longer in the list. The next method that will be searched is enumerated.

Fig. 6 shows the main screen for keyword search. The top window displays messages to the user. The Keywords-in-Database window displays the keywords in the database. There are more than 3000 keywords for the UNIX tools database used in the study. Below the keywords window is a menu of operators that a user can select with a mouse to construct queries. The List of Previous Queries window displays the queries searched so far. The window below it displays the items retrieved by the currently selected query. The Current Query Form window on the bottom is where the users enter queries. Queries use Boolean operators.

Fig. 7 shows the main screen for enumerated search. In enumerated searching, the user traverses the classification tree for the database. The top window displays user messages. The Current Root Class window displays the node in the classification tree where the user is currently positioned. In the window below that, the subclasses are displayed. The Super Classes of Current Root Class window displays the path through the tree of classes that was traversed to reach the current root class. The Description of Current Root Class window displays a textual description of the current root class. This window is also used to display the parts in the leaf nodes, when the “Show Parts” button is clicked.

Fig. 8 shows the main screen for faceted classification. We implemented the faceted interface with a spreadsheet. The top window is for user messages, as in the other methods. The next window contains the facet names, one per column. Below this are the terms in each of the facets. By middle-clicking
on a term, the user can see a list of synonyms for the term. Other windows provide the ability to sort the terms within a facet, to center the spreadsheet on a given term, and to retrieve information about the current part.

Fig. 9 shows the main screen for the attribute-value method. The top window, as before, is for user messages. The Attributes Available window lists all attributes available for the database. When the user clicks right on an attribute, it becomes the current attribute. The Values of the Attribute window lists all values for the current attribute. A search is specified by selecting sets of attribute-value pairs by clicking on values for the attributes. Selected attribute-value pairs are added to the Pattern to Match window, and multiple attributes can be used in a search. They can be combined with an “And” or an “Or.” The Parts that Match Pattern window shows that two parts, SUM and TAIL, match the selected pattern.

We attempted to provide interfaces for the four methods that were as consistent as possible to minimize the impact of different interfaces on the result. For example, each of the four methods uses the same set of screens to display information about components.

B. Proteus Interface

The success of Proteus as a vehicle for the evaluation experiments is dependent on the quality and consistency of the user interface across representations. It might be argued that using a consistent interface could discriminate against a given representation method if the method was best suited to an interface of a given type. However, there is currently no evidence to support this view, and thus a common look and feel for all methods was used for the experiments.

We used a joint application design (JAD) [27] to design the major features of the interface. In a JAD, users and developers meet and design an interface using some rapid prototyping method. Our user group was drawn from several member companies of the Software Productivity Consortium who were interested in reuse library technology.

C. Proteus Development

Proteus consists of 22,000 lines of source code written in Common Lisp and the Common Lisp Object System (CLOS). This includes the library mechanism and its graphical user
interface, the data files used to define the libraries’ databases, the tools used to collect information during the experiments, and the report generators used to present the results from the automated portion of the experiment’s data collection. Proteus was developed on an Apollo Domain 3000, and then was ported to a SUN 4 workstation for this study.

Table III gives timings (in s) for Proteus operations on the Apollo (times on the Sun 4 were on average 65% faster). Load means starting a given representation method, searching

is the time needed to locate and retrieve items in response to a query, and display is the time needed to display the retrieved items. Empty cells in the table indicate that negligible time is required for the operation. For example, faceted and enumerated searches have essentially no retrieval time associated with them, because of the way they are structured. We feel that time differences between operations for different representations did not bias the experiments.

D. Test Collection

The test collection used in the experiment is based on one by Maarek [24]. We modified it by selecting a subset of the queries in the original collection, and arranging them in four sets of seven queries. The following queries are examples of those used in the experiment.

- Which UNIX tools help me locate a regular expression in a file?
- Which UNIX tools help me evaluate the similarities between two files or compare them?
- Which UNIX tools help me remove or delete a file?
The set of parts in the test collection was changed somewhat, because we were using a different version of UNIX than Maarek. Our collection consisted of 120 UNIX parts, primarily from Section 1 of the UNIX manual. We obtained new relevance judgments for the queries from two UNIX experts. Relevance judgments determine which items in the database satisfy the query. The process that we used in getting our test collection is summarized in Fig. 10. Once we established the parts we would use, they were indexed using four methods: attribute-value, enumerated, faceted, and keyword. The size of the collection (120 components) is typical of the sizes of asset collections today, and thus provides good generalizability of the results.

VI. ANALYSIS OF EXPERIMENTAL RESULTS

A. Searching Effectiveness

Searching effectiveness refers to how well a given method supports finding relevant items in a database. We have evaluated searching effectiveness with recall and precision, the traditional measures, and with overlap.

B. Recall

Recall is the number of relevant items retrieved over the number of relevant items in the database. We show the recall performance of the four representation methods using boxplots (see appendix) in Fig. 11. The plots show that recall values for all four methods varied between 0 and 1, with median values in the high 30% to low 40% range. The plots indicate that there is no significant difference between the methods in terms of recall.

An analysis of variance for recall, shown in Table IV, failed to reject the null hypothesis that representation method does not significantly affect precision at the 0.05 α level, and confirmed that no significant difference exists between the four methods.

C. Precision

Precision is the number of relevant items retrieved over the total number of items retrieved. The following boxplots show the precision performance of the four representation methods. The plots show that precision values for all four methods varied between 0 and 1, with median values varying from about 0.5 to 1. The plots indicate that there is a significant difference between the methods in terms of precision.

An analysis of variance for precision (Table V), however, failed to reject the null hypothesis at the 0.05 alpha level, and we thus conclude that no significant difference exists between the four methods.

D. Overlaps

Previous experiments with document collections have shown that though the differences in recall and precision between representation methods may not be significant, different methods will retrieve different documents [22]. This is an important consideration in the design of retrieval systems, strongly implying the need for multiple representation methods. The reasons for this phenomenon have not been systematically studied, but are somewhat explained by the fact that different methods have different indexing terms for the same item.
We examined this possibility by calculating and comparing overlap measures for the four methods. The overlap measure, \( X \), we used is defined as follows:

\[
X = \frac{|m1 \cap m2|}{|m1 \cup m2|},
\]

where \( m1 \) is the set of documents retrieved by one method and \( m2 \) is the set of documents retrieved by another method. This measure, then, is the ratio of the number of relevant documents in the intersection of two methods divided by the number of relevant documents in their union.

The overlap measures in Table VI range from 0.72 to 0.85. This shows that the methods do indeed retrieve different relevant items, though their recall and precision figures are not significantly different. These overlaps are much higher than those reported for document collections.

**E. Searching Time**

The boxplots below give user searching times for the four methods. User searching time is the length of time, in minutes, spent by a user to search all of the queries for a given method. There are substantial search time differences between the methods, and Duncan post-Anova tests [26] confirmed that the differences between all the methods are significant, with a difference of 18.4 min (60%) between the slowest method, keyword, and the fastest method, enumerated.

**F. UNIX and Programming Experience**

We tested the hypothesis that UNIX and programming experience would be significant predictors of searching performance and found that they were not. The correlations can be seen in Table VII.

None of these correlations is significant. The lack of significant correlation between programming and UNIX experience, and searching effectiveness and search time, may be caused by the fact that most of the subjects had such experience (see Fig. 4). In one case in the study where a subject had no such experience, the subject reported finding the searching task very confusing. It appears that a certain minimal level of knowledge of UNIX and programming is required, though our sample had too few subjects in this category for this to appear statistically.

**VII. SURVEY RESULTS**

We asked each of the subjects questions about the search methods and about Proteus to get additional insight into the usefulness of the methods. The analysis of their responses follows.

**A. Preference of Subjects for the Methods**

Subject preference for the methods was obtained in two ways: by asking the subjects to rank order the methods, and by asking the subjects to rate the methods on a 7-point scale.

---

**TABLE VI**

<table>
<thead>
<tr>
<th>Method</th>
<th>Attrib.-value</th>
<th>Enumerated</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerated</td>
<td>.72</td>
<td>.65</td>
<td>.79</td>
</tr>
<tr>
<td>Faceted</td>
<td>.85</td>
<td>.79</td>
<td>.76</td>
</tr>
<tr>
<td>Keyword</td>
<td>.70</td>
<td>.79</td>
<td>.72</td>
</tr>
</tbody>
</table>

**TABLE VII**

<table>
<thead>
<tr>
<th></th>
<th>UNIX Experience</th>
<th>Programming Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>.16</td>
<td>-.01</td>
</tr>
<tr>
<td>Precision</td>
<td>.16</td>
<td>-.02</td>
</tr>
<tr>
<td>Search Time</td>
<td>-.18</td>
<td>.16</td>
</tr>
</tbody>
</table>

**Fig. 13** Boxplots of user search times for the methods.

**Fig. 14** Subject ranking of representation methods.

**Fig. 15** Subject rating of representation methods.
The results of this study were consistent with the findings of documentation retrieval experiments. This indicates that the findings of document retrieval experimentation may well be applicable to the reuse representation problem.

- All methods were ranked best and worst. This shows that no method was regarded as best, or even as adequate, by all subjects.
- Subject programming and UNIX experience had no significant effect on searching performance as measured by recall, precision, and search time.
- There was no significant difference between methods for helping subjects understand components.
- A library system can be designed to support multiple methods. Proteus shows the technical feasibility of this.

IX. ADVICE TO PRACTITIONERS

An experiment must be replicated many times under varying conditions by independent researchers before its results are generally confirmed. Nevertheless, because many people are now constructing reuse libraries, and because it is unclear when or if replications of this study will happen, we offer the following advice to practitioners based on our results.

- Represent your collection in as many ways as you can afford. None of the methods is sufficient for finding all relevant components for a given search. Having more representations will increase the probability that relevant items will be found. Also, individual users do better with, and prefer, one method over another, and are individualistic in their choices. We do not currently understand the factors that cause these differences in subjects.
- If you must choose just one method and are primarily interested in searching effectiveness and cost benefit, choose free text keyword. It is best in terms of cost-benefit ratio for searching effectiveness, because all methods are equally effective, and free text keyword is the least expensive, because it does not require human indexers.
- If you must choose just one method and are primarily interested in search time, enumerated is best.
- Do not expect any of the methods to adequately support understanding of the components. The methods were rated equally for helping users understand components, and none of the methods did better than moderately well. This is an area where other techniques like domain analysis may be needed.

These results have already proven useful to practitioners building reusable asset libraries such as Mountainet [28] and InQuisix [29].

X. FUTURE WORK

The most important future work is to replicate this study with other subjects, different or larger databases, different user interfaces, different indexers, different domains and domain experience, and other representation methods. It would also be valuable to rerun the experiment with subjects who had more experience using the methods. Such replications are necessary,
as they are with all scientific experiments, to strengthen the validity of the findings reported here. It is also important to look at the questions used in this study using alternative experimental methods, e.g., a task-based experiment in which subjects would build a system using reusable components.

It is clear from the experiments that subjects varied significantly in terms of searching behavior and preferences for the methods. There is currently no psychological theory that allows these differences to be predicted based on known attributes of subjects. This is an interesting area for long-term basic research.

Current estimates of costs associated with representation methods are based on first principles. We know, for example, that free text keyword searching is generally less expensive than controlled methods, because it requires no human indexers. More precise data on costs associated with the methods, based on empirical observation, should be gathered.

The component understanding problem is very important and was addressed directly in this study only by one survey question. More data on the effectiveness of the methods for analysis and understanding of components and systems should be gathered.

To move a tool like Proteus from a laboratory to a field environment, it should be rewritten in a production-oriented language, e.g., C++. This change would allow Proteus to be available on the maximum number of platforms, and, because of the greater number of C++ developers in comparison to Lisp/CLOS developers, would minimize cost of maintenance and uprating of the Proteus tool.

APPENDIX

BOXPLOTS

A boxplot [30] (see Fig. 17), is a graphical representation of the distribution of a set of data. The bottom of the box is the 25th percentile and the top of the box the 75th percentile. Thus, half of the data points in the set fall within the box. The difference between the 75th and 25th percentiles is the midspread. The top hinge is the largest data point less than or equal to 1.5 midspreads above the 75th percentile; the bottom hinge the smallest point greater than or equal to 1.5 midspreads below the 25th percentile. An outlier is any data point above or below a hinge, but no more than 3 midspreads above or below. An extreme outlier lies more than 3 midspreads from the hinge.

The line across the middle of the box is the median, or 50th percentile. The shaded area around the median is the 95% confidence interval for the median. It is placed symmetrically around the median according to the following formula:

\[
\text{median} \pm (1.57 \times \text{midspread} \div \sqrt{n})
\]

In this formula, \(n\) is the sample size. If the shaded areas in boxplots for two different variables fail to overlap, it indicates that the medians for the two variables are significantly different.

ACKNOWLEDGMENT

Many people contributed to this study. Y. Maarek of the IBM Thomas J. Watson Research Center provided the basis of the test collection. R. Prieto-Díaz supplied the faceted classification of the UNIX components. L. Escobar of LSU, D. Lubinsky and D. Weiss of AT&T Bell Laboratories, and S. Davis of Clemson reviewed the experimental design and improved it with their comments. D. Lubinsky also reviewed the data analysis. J. Kirby participated in an inspection of the experimental data, and with S. Wartik, R. Bechtold, and J. O’Connor reviewed the technical report. G. Wheatley provided programming support for the development of Proteus. Members of the SPC reuse technical advisory group provided valuable feedback on Proteus and the experiments.

REFERENCES


William B. Frakes received the M.S. degree from the University of Illinois at Champaign-Urbana, USA, and the M.S. and Ph.D. degrees from Syracuse University, Syracuse, NY, USA.
He is an Associate Professor and Director of the computer science program at Virginia Polytechnic Institute and State University, Falls Church, VA, USA. Previously, he was Manager of the Software Reuse Research Project in the Software Productivity Consortium, and Supervisor of the Intelligent Systems Research Group at AT&T Bell Laboratories. Dr. Frakes is author of Software Engineering in the UNIX Environment (Prentice-Hall 1991), and Editor of Information Retrieval: Data Structures and Algorithms (Prentice-Hall 1992) and Advances in Software Reuse (IEEE Press 1993). He edits ReNews, an electronic software reuse newsletter, and ACM SIGIR Forum. He is Chair of the IEEE TCSE subcommittee on software reuse.

Thomas P. Pole received the B.S. degree in computer science from the University of Maryland, College Park, MD, USA, and the M.S. degree from Johns Hopkins University, Baltimore, MD, USA.
He is currently working with Advanced Decision Systems, a division of Booz-Allen and Hamilton, Inc., Vienna, VA, USA. His research interests are in software engineering tools and methods, specifically software reuse, object-oriented methods, and component-based software development (the generation of software systems from domain-specific specification languages that build upon collections of domain-specific reusable components and architectures).