Experiences with a Faceted Classification Scheme in a Large Reusable Software Library (RSL)

Jeffrey S. Poulin
Kathryn P. Yglesias

International Business Machines Corporation

Abstract

This paper presents experiences with software classification in a large corporate reuse software library (RSL) at IBM. We use facets extensively as one method of component classification in the IBM RSL. However, facets alone cannot adequately provide all the information needed to fully classify and understand a reusable component. Experience with an operational RSL reveals that we require a combination of classification techniques to meet the needs of software developers. Following an overview of the IBM classification method, we discuss the issues surrounding the use of facets and software classification in a large reuse system and give techniques used at IBM to address those issues.[1]

1.0 Overview

Storing, searching, and retrieving software from a repository of reusable components is central to the practice of reuse. Each of these activities relies on the existence of a systematic method of organizing the components so reusers can match existing reusable parts to their current needs. Classifying software allows reusers to organize collections of components into structures that they can search easily.

An RSL uses a classification scheme to create an index to assist in the physical storage of components in the library (or database) and to provide input to search tools. The method of classification is an important ingredient in determining the types of indices that can be used, the types of searches that can be conducted, and the types of tools that reuser can or must use. The method of classification also determines the accuracy of possible searches and the precision of the results [20].

However, classification requires an investment in resources. Someone must create and maintain the classification scheme and all instances of the scheme. Fast, interactive searches can require large and complex indices not necessary in other situations. Automated indexing methods, manual classification, the nature of the reuse repository, and the types of available retrieval tools all influence the classification method.

We use the popular method of faceted classification extensively in the internal IBM RSL. A facet term describes a key aspect of the software; each facet has a related set of terms to describe the possible values of that facet. Selecting an appropriate component consists of matching facet terms of software in the RSL to a specific need. However, facets alone cannot adequately provide all the information needed to fully classify and understand a reusable component. We find that we require a combination of classification techniques to help software developers locate, assess, and integrate reusable components into their products.

This paper presents experiences with software classification in a large corporate reuse library system at IBM. An overview of software classification precedes an explanation of the IBM reuse system. We then discuss issues surrounding the use of facets and some techniques we use to address those issues. Following the IBM software classification experiences we discuss a series of enhancements to software classification required in a large production environment.

---

[1] Presented at the *Seventeenth Annual International Computer Software and Applications Conference (COMPSAC'93),* Phoenix, AZ, 3-5 November 1993, pp. 90-99

2.0 Software classification

Systematically ordering software into classes that specify the allowable uses for the software is particularly complex. Unlike the specific functions provided by computer hardware, software often possesses an overall ambiguity and generality that is difficult to express. There has been general success identifying math and I/O routines. However, although the software community understands simple utilities and abstract data types very well, this understanding becomes lost when the software deals with more abstract ideas and algorithms containing intermingled functions and side effects.

Various methods to classify software have been proposed and implemented, including numerous formal and automated techniques. However, most of the methods actually in practice are based on classification lessons learned in library science. Of these methods, there are four major categories [5]:

1. Enumerated
2. Attribute-value
3. Facets
4. Free-text

In *Enumerated* classification parts are organized into classes. These classes are usually hierarchical; an example of enumerated classification is the Dewey Decimal system. An sample search in an RSL that uses enumerated classification might go as follows: to find a routine for solving third order differential equations, you first look for Math Routines, then Calculus Routines, then Differential Calculus Routines. You would then examine a list of available routines. Searching for software is analogous to looking something up in the table of contents in a book. Enumerated classification has the advantage that it is well understood by most people and therefore easy to use. However, enumerated classification has the following disadvantages:

- the index must be built manually, a costly and error-prone process,
- the ambiguity of a part can cause it to fit several places in the scheme, which can make it difficult to locate, and,
- the structure is not easily balanced, which makes it awkward to use if there are many parts of one type and few of another.

In *Attribute-Value* classification parts are described by a set of attributes and their values; e.g., if the attribute is "Author_name," the value might be "John Smith." The attribute can take any value assigned by the person classifying the part. Locating a part consists of specifying an exact value or range of values for a subset or all of the attributes. This method is easy to use and can be partially automated, such as for the attribute "Object_code_size." However, it requires a more sophisticated search mechanism and incurs relatively poor performance during searches. Attribute-value also suffers the same ambiguity problem as does enumerated classification. Without some way to control the attribute values, reusers can use different terms to describe the same part, thereby making it difficult to find the part. Some libraries implement synonym lists or a thesaurus to address this problem.

In *Faceted* classification parts are described by a set of terms, or facets, and facet values. In this way facets are similar to the attribute-value method. However, with facets the choice of values is limited. This eliminates the problem of ambiguity in deciding the best value for a term or attribute. For example, if the facet is "Operating_System," allowable values might be one of (AIX, VM, MVS, OS/2). Because facets limit the choice of terms, search performance can be very good.

In *Free Text Keyword* classification parts are described by English words or phrases. The text can be entered by a person who is classifying the part or can be extracted automatically from the documentation or source files [13].

Library science has two general strategies for handling free text indexing. These are through the use of *controlled* and *uncontrolled* vocabularies. Three of the four methods for classifying software; enumerated, attribute-value, and facets, are controlled vocabulary techniques. Free text, however, can be either uncontrolled or controlled. Free text is uncontrolled when terms are either chosen *ad hoc* by the person classifying the part or are automatically extracted from text, either with or without syntax. Free text is controlled when keywords are matched to a synonym list or thesaurus. Free text is easy to implement and use but suffers from difficulty in matching requirements to existing software unless, like attribute-value, the RSL implements a more sophisticated search tool.

A study on the relative recall,[2] precision,[3] search times,[4] overlap,[5] and user preference of each classification method shows that each method has strengths and weaknesses, and therefore a RSL should implement as many as practicable [5]. In fact, IBM uses all four techniques in the organization of the IBM corporate reuse environment. We structure the environment into inventories and libraries using enumerated techniques, exploit free text in searches, and classifying individual reusable units with facets and attribute-values.

2.1 Overview of the IBM Classification Scheme

The IBM reuse environment consists of high level groupings of related libraries called *inventories.* We use inventories as the entry point for the reuser in the environment. Example inventories include "documents," "commercial software," and "federal software."

Inventories contain one or more *libraries.* Libraries further enumerate the options available to the reuser. Examples of libraries within the "federal software" inventory are "Ada collection packages," "flight simulation software," and "aerospace navigation software."

Libraries consist of collections of *components.* A component is a bundle containing all the information needed to ensure a part can be reused efficiently; the component is our basic unit of reuse [9]. Figure 1 shows how libraries consist of components and how components consist of elements of information. The information elements can be reusable (such as a source code file) or can simply be information elements, such as "integration instructions," intended solely to assist the reuser.

Note that classification involves information used specifically for searching for reusable components. The other elements of information are required to assist reusers evaluate and reuse software; those information requirements are not included in this discussion. However, in some cases elements originally intended only for informational purposes become part of a search mechanism. For example, short (less than one page) free-text abstracts of reusable components are very useful for understanding a reusable component. Although abstracts are not used in faceted classification, some search tools, including the IBM tool, use abstracts to build indices of the reuse library or scan the abstract during the search process.

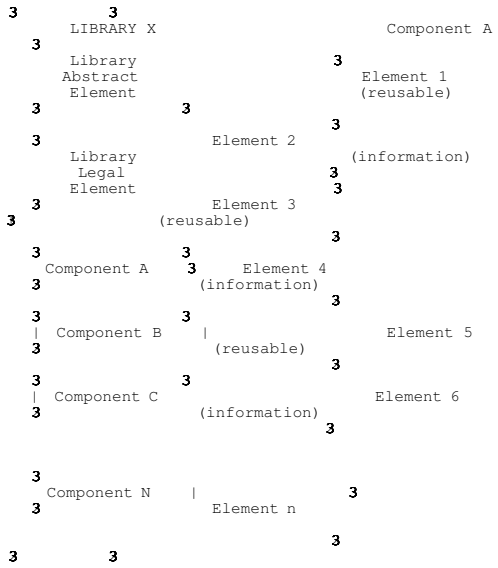

Figure 1. Libraries, components, and elements. Libraries contain related components. Components contain related elements. Elements contain either reusable parts or information about the reusable parts.

Facets and attribute-values are together called *classifiers.* As shown in Figure 2, classifiers describe libraries, components, and elements [8]. The use of both methods rather than the exclusive use of facets was one of the earliest issues addressed when developing the IBM software classification scheme. The need to track non-enumerable data such as individual author name and the need to track data with potentially large ranges of values such as department codes mandated the use of attribute-value classifiers.

---

2 Recall is the number of relevant items retrieved compared to the total number of relevant items in the database.

3 Precision is the number of relevant parts retrieved compared to the total number of parts retrieved.

4 Search time is the elapsed time required to locate an applicable reusable part.

5 Overlap is a measure of the total unique parts retrieved by each classification method.

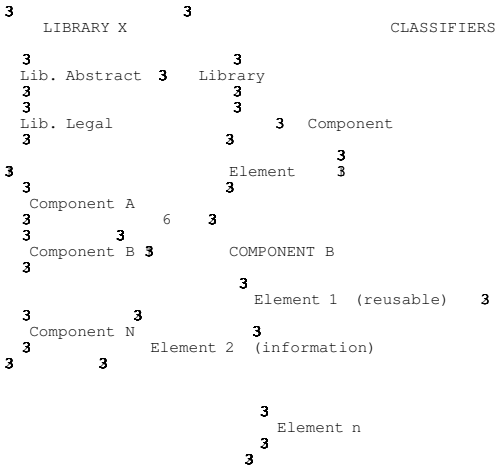

Figure 2. Libraries, components, elements and classifiers. Classifiers describe libraries and components. We use classifiers to identify which libraries to search and to locate and retrieve potentially reusable components.

2.2 The origin of faceted software classification

In January 1987 Prieto-Diaz and Freeman proposed a faceted classification scheme for software [15], [16]. Recognizing that previously existing methods of organizing software were inadequate for large, continuously growing libraries, they proposed a component description format based on a standard vocabulary of terms. This faceted approach was more descriptive and extensible than earlier enumerative methods, and included a conceptual closeness graph of related facet terms to aid in matching user needs to stored components [17].

Table 1. Prieto-Diaz and Freeman faceted classification schedule

Facet	**Description**	**Examples**
Function	Specific function performed by the component.	add, delete
Object	What the component acts on.	arrays, files
Medium	Where the action is executed.	buffer, tree
System type	Functional or application independent area.	compiler, scheduler
Functional area	Application dependent activity.	budgeting, DB design
Setting	Where the application takes place.	advertising, finance

The Prieto-Diaz and Freeman approach centers on the six facets shown in Table 1 and seeks to provide a preliminary schedule intended for functionally identifiable software products ranging from about 50 to 200 lines of code.

2.3 Issues with a faceted taxonomy

Although many RSLs use the faceted taxonomy, many implementation issues need to be addressed. Our experience shows that the following issues are the most significant:

2.3.1 Consistency

Keeping the classification consistent requires a thorough understanding of the domain or domains which the classification covers. The first step in understanding a domain requires working with domain experts and users of the classification to understand their use of terms. The analyst must then look for conceptual similarities and differences among the terms of the various user groups.

The analyst seeks to create a set of terms at as similar a conceptual level as possible by including variations on a shared concept within a single “group” of terms. The analyst can help reusers by grouping related terms using one of several techniques, including conceptual closeness [15], [6], lattices [4], or thesauri [2]. IBM uses a synonym matching tool to help unify terms into consistent structures identified by a single key, or *root* term. The root term identifies a set of closely related terms that are either synonyms or related by their use in a particular domain.

Because every user group has a select area of specialization, small differences of concept become very significant. For example, the first IBM classification schedule had numerous terms related to operating systems and aerospace because IBM does a lot of work in these areas. However, even within these domains a term would have several interpretations. For example, a developer in the MVS operating system will disagree on terms used to describe a memory-mapping concept also used in the VM operating system. In these situations the classification analyst must look for where subtle differences in a term will compromise the consistency of the rest of the classification scheme. Segregating the vocabulary into groups identified by root terms increases the reuser's chances of locating an applicable component.

2.3.2 Brevity

Brevity is one of the most important considerations when defining a classification scheme; the analyst must keep the set of facets and the terms in each facet as brief as possible. However, a diverse user group with many different views of the salient characteristics of a given domain make this difficult to achieve. For example, the original classification scheme at IBM included all of the attributes which Grady Booch used to define his Ada Abstract Data Types (ADTs), since this was one of the first commercially available RSLs [1]. Eventually we condensed many of the Booch attributes by aggregating related functions, characteristics, and features into single facets and then grouping these ADT-unique facets under an ADT domain.

2.3.3 Resolving ambiguous terms

Ambiguity of terms arises when more than one group of users will retrieve parts from the RSL. Developers of operating systems and developers of office automation systems both refer to the term *address*, which has two completely separate meanings. In this case we can resolve the ambiguity fairly easily, but other domains have much greater overlap and so require a much more detailed analysis.

Our guidelines to resolve these situations state that the same term cannot exist as a root term within the same facet, even across domain boundaries. This guideline helps decrease user confusion and simplifies implementation of the RSL. User confusion occurs when the user considers several domains as good candidates for finding parts but each domain contains the same term with different meanings. An example is the term *cursor* for the "Object" facet in the graphical user interface (GUI) and operating systems domains. In a GUI, *cursor* refers to a symbol on the output device, whereas in the operating system domain *cursor* refers to a type pointer.

If the ambiguous term is significant in both domains, we resolve the situation by making it a synonym of a root term which developers identify within the domain as conceptually similar. The decision on which domain will maintain ownership of the root term usually depends on the existence of an appropriate alternative in the alternate domains. In the case of *cursor*, it was retained as a root term for the "Object: user interface" facet (with a synonym of *mouse pointer*) and *pointer* became the root term for "Object: data structure."

2.3.4 Extensibility

Any classification scheme must be able to adapt to new requirements. The addition of new domains affects the classification structure by potentially requiring new facets and the need for new terms in existing facets. The most difficult aspect of extensibility involves controlling the ambiguity and brevity of the classification scheme while being responsive to user needs and concerns.

A decision to limit addition of terms (and control brevity) made on a purely conceptual basis may not be acceptable to one or more user sets whose terms are being condensed or grouped. We must use experience to weight the factors and arrive at a solution. These factors can include:

- willingness of the groups to accept terms,
- the number of proposed changes,
- frequency of use of the RSL by the groups,
- number of users in each group, and
- cohesiveness of each group.

2.3.5 Maintaining facets and terms

The dynamic nature of programming requires constant maintenance of not only software but also of the terms needed to describe it. For example, we changed the facet "Function" to "Function (What it does)" because many people confused the intended meaning with several other possibilities. We also changed the facet "Proven Operating System" to "Software Environment" to accommodate network operating systems and other environments (such as IBM's Customer Information Control System (CICS)) which are not technically operating systems.

2.3.6 Balancing administrative and technical needs

Technical users frequently comment that they only want a few simple facets (e.g., "their" facets and terms). They request simplification of the classification schedule by removing all facets that they feel do not apply to them. In fact, the facets they target for removal are usually the administrative classifiers which identify the library, owner, legal, and other restrictions related to reusing the component. Our experience shows we must include these administative classifiers and information elements to provide the necessary information to all

organizations affected by the component: designers, testers, maintainers, legal reviewers, etc. However, part suppliers and reusers can find the wealth of information surrounding a reusable component somewhat overwhelming.

3.0 Experiences with the faceted taxonomy

Early work in faceted classification provided the architectural basis for the IBM taxonomy [15]. We redesigned an existing prototype reuse system to address needs of an expanded audience and range of life cycle products. We then implemented the faceted taxonomy in the new tool. Since establishing the initial set of classifiers, we have conducted two major classification evaluations and re-designs. Furthermore, we made approximately forty operational updates to the classification to incorporate changes requested by nearly constant user feedback.

3.1 The first set of IBM facets

IBM based the first set of facets on the work done for the STARS program. Many of the same individuals working on the IBM STARS team also worked on the IBM RSL. However, due to differences in the audiences and goals of the RSLs, the resulting classifications and RSLs are completely different tools.

The best feature of the first set of IBM facets was the close mapping of the classification to one of our major libraries, Booch's Ada abstract data types. This strength rapidly disappeared as we encountered difficulties due to terminology differences in other ADT libraries as well as the addition of avionics part sets from IBM's Federal Systems Company. The avionics library included math models, ADTs, and fairly large reusable components for navigation and guidance.

Agreement on the first set of facets was difficult due to the need to expand the scope of the library. For example, we did not have adequate examples of how to classify user documentation, designs, or test cases. Consequently, the first set of facets included internal (process) and external (customer) documentation, but did not attempt to cover the other software life cycle products. We reviewed the classification quite extensively, but until recently we did not have sufficient experience to determine the adequacy of the scheme. Although few changes have been requested, we continue to conduct research in this area.

3.2 Current set of Classifiers

The current set of IBM classifiers resulted from the issues and experiences described above. These classifiers have proven to work across all phases of the software lifecycle. They have also proven flexible enough to describe and discriminate between parts developed by contractors, vendors, and IBM organizations over the wide diversity of environments in which we operate. Despite these successes, user feedback alerts us to a continuing need to analyze usability and retrieval precision [20].

A partial list of the IBM classifiers appears in the following table; note the evolution from the original set of facets defined by Prieto-Diaz and Freeman. It is also interesting to note that IBM's experience has led to a classifier set with similarities to those of the other companies or groups shown in Section 4.0, and to proposed standards such as the current work by the Reuse Interoperability Group (RIG), an organization originating from the Software Technology for Adaptable, Reliable Systems (STARS) program [18].

3.3 Further enhancements

Even with IBM's involvement with formal reuse for many years, the ability to satisfy the reuse customer (for example, dealing with challenges offered by adapting faceted classification to object-oriented environments [10]) requires continuous evolution of our tools and methods.

Table 2. Partial listing of the IBM classifiers

Facet	Description	Examples
Algorithm	Technique to perform an action.	bubble, merge
Application Domain	Broad area of application.	advertising, aerospace
Certification Level	IBM quality rating.	Certified, as-is
Component Size	Component size in LOC or words.	512 LOC, 8245 words
Data Structure Characteristics	How the data object is implemented.	bounded, directed
Development Standards	Compliance with standards.	ISO 9000, DoD-2167A
Function	What the component does.	sort, add
Object	What the component acts on.	buffer, array
Implementation	Describes various techniques.	sequential, dynamic
Implementation Language	Programming language.	C++, Ada
National Language.	National language	English, French
Proven Hardware	Tested computer platforms.	R/6000, S390
Proven O.S.	Tested Operating Systems.	AIX, OS/2
Support Level	Guaranteed service provided.	Limited, Full

3.3.1 Integration with text search techniques

Although the extensive list of classifiers provided a complete descriptive scheme, customers requested the ability to execute key word searches in addition to classifier searches. There were several reasons for this. Groups with limited domains are often familiar with the parts available for their domain and choose to access them directly by appropriate keyword search. Other groups find the time required to understand the classification a luxury which they cannot justify (a real example of short term goals off-setting long term benefits).

We encourage reusers to use the faceted classification to locate feasibly usable components from a broad set of libraries, then to iterate on the search results through use of keyword search. This two-fold approach helps control search costs and provides options for very specialized winnowing of the first order selection.

3.3.2 Hierarchical facets

To further reduce the number of classifiers initially shown to the reuser, we defined a hierarchical ordering of existing classifiers. By grouping the facets on which users did not want to focus (such as administrative facets), the programmer can more quickly re-define the correct search scope as his requirements evolve. Because of the inter-dependence among facets, the hierarchical structure is not pure. The three-dimensional model which reflects the actual dependencies of the independent views (e.g., facets) must be simplified and to some extent hidden from the user by the RSL's interface. Hierarchical facets are one of the most effective ways found so far to help accomplish this goal.

3.3.3 Domain requirements

Despite extensive domain research and analysis, no set of classifiers will satisfy all users. Development organizations with unique products or computing environments demand the ability to create custom sets of classifiers for use by their organization. Differences in jargon and culture further complicate customer needs. As a result, we adopted a mechanism to allow selected user groups to add facets for local use.

3.3.4 User perspectives

We face a fundamental problem in classification in that reusers often have different understandings of the terms. The cognitive processing of the potential reuser often causes the reuser to develop a concept of the problem in such a way as to allow for a variety of potential solutions. This tendency to think in terms of the “solution space” rather than the “problem space” means the reuser will fail to consider potential solutions which fall outside the scope of the solution *as he sees it.*

The nature of language and classification for reuse causes interpretation options. Once the analyst has completed the classification, the responsibility to interpret the classification and think in the problem space shifts to the reuser. Education on classification and reuse must accompany the installation of a RSL. Furthermore, the education must extend beyond use of the features of the RSL, it must provide the potential reuser

with the ability to flexibly evaluate a problem and find full or partial solutions. The following experience illustrates the importance of reuse education in determining the appropriate use of Abstract Data Types (ADTs) developed for reuse.

One of our development projects reported a performance problem with a *list* ADT from our RSL which they chose to reuse in their code. Our technical consultant investigated with questions related to their abstraction requirements rather than the specifics of the alleged performance problem. He found that the abstraction required a direct access method, multiple iterations, fast sequential access, fast insertion, and fast key-based retrieval. He recommended that the project use an *AVL-tree* rather than a *list*. With this choice of ADT, the performance problem went away.

3.4 When to apply facets

The use of facets depends on the situation. The previous sections have discussed some of the benefits and some of the limitations of use of faceted classifications. IBM's reusers are a heterogeneous set of developers located around the world. They develop software ranging from operating systems to business applications to medical systems and write in programming languages ranging from COBOL to C++ to Assembler. The needs of such a diverse group require a mechanism such as a faceted classification.

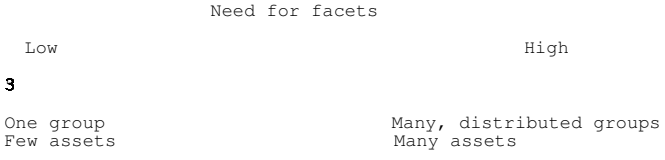

Figure 3. When to use facets

Few other companies have as extreme a set of programming requirements as IBM. As the set of requirements are scaled down — smaller groups of programmers, fewer languages and fewer domains, the requirements for a faceted classification decrease. At some point, the maintenance requirements and overhead for retrieval cause facets to become a detriment.

Remember that we use facets to provide a systematic method for focusing on the *problem space* rather than the *solution space*. This means that a programmer can use code written for an entirely different application, often even in a different language, when the programmer focuses on what the code accomplishes (and its performance characteristics) and not on a preconceived solution. Finding these opportunities for reuse depends on a reliable classification method and RSL.

4.0 Related Work

We limit the following discussion to those techniques based on library science. However, library science is not the only field contributing to classification research. For example, the mathematical field of category theory has established methods to characterize algebraic structures. Formal specification languages using relation algebras can formally specify the behavior of software components [12]. Denotational semantics and predicate calculus specifications consisting of preconditions and invariant assertions can also provide accurate, provably correct descriptions of software [7]. However, most operational libraries rely on principles adapted from library science and use the four basic classification methods described earlier in this paper.

Facets have proven effective in organizing information in numerous disciplines, such as zoology. Within computer science, facets have been used to classify documents for reuse [11], to organize and retrieve appropriate programming tools [14], and to group, analyze, and predict errors in software development [3], [19].

Table 3 contains a sample of operational reuse libraries, the organizations which sponsor or developed them, and the classification method used by each.

Table 3. Sample Reuse Libraries

Library	**Developer**	**Classification Method**
Catalog	Bell Labs	Free text
Asset-CARDS	Dept. of Defense	Faceted
REBOOT	Europe	Faceted
Reuse	Texas Instruments	Free text (controlled keyword)
RAASP	Westinghouse	Enumerated Faceted
RLF	Unisys	Faceted
Asset Library	GTE	Faceted
RSL	Intermetrics	Free Text (uncontrolled keyword) Enumerated
CAMP-PES	US Air Force	Enumerated Attribute-Value

5.0 Future Work

The dynamic nature of our development environment demands ongoing maintenance and enhancement of the classification schedule so we can remain responsive to customers. In addition, all corporate practices and guidelines must include the standards for software classification and reuse for them to become fully ingrained in the development process. Identifying affected areas, including new technologies which overlap with reuse (such as object oriented), ensures that we continue our progress.

As we build our sets of reusable parts into new domains and languages, we must expand the set of classifiers to embrace the new concepts and terms. We constantly apply what we have learned in our early classification experience to new areas. These include defining tool requirements to ease part retrieval and developing standards to support packaging and storing of reusable parts.

One area of future work involves the level of abstraction in the current set of facets and terms. We must provide the appropriate abstractions to conduct optimal searches of class libraries containing a wide range of methods and features. This conflicts with the desire to reduce the overall number of facets for general usability, since we tend to add new facets and classifiers as we add new functions and classes to the RSL.

6.0 Conclusion

A faceted taxonomy applies best when many diverse, geographically distributed organizations need to share a large set of similarly diverse software. Under these conditions, facets provide one vehicle to normalize terminology and programmer predispositions to a particular solution. Facets present a systematic approach to the *problem space*, thereby providing opportunities to locate and reuse parts which might otherwise not contribute to a solution.

However, facets cannot satisfy all classification requirements. Other techniques must also contribute to the knowledge base, including attribute-values, enumerated, and free-text. Users demand flexibility and require more options. The classification analyst must balance the needs of the various user groups and provide a meaningful, accurate, and useful classification standard. Failing to moderate the numerous demands for information and special requirements can result in overwhelming the user with information.

7.0 Cited References

[1] Booch, Grady. Software Components with Ada: Structures, Tools, and Subsystems. Benjamin Cummings, Menlo Park, CA, 1987.

[2] Burton, Bruce A. et. al., "The Reusable Software Library," *IEEE Software*,, July 1987, pp. 25-33.

[3] Chillarege, R., et.al., "Orthogonal Defect Classification- A Concept for In-Process Measurements," *IEEE TSE*, Vol.18, No.11., November 1992, pp. 943-956.

[4] Eichman, David and John Atkins, "Design of a Lattice-Based Faceted Classification System," *Proc. of the Second Int. Conf. on Software Engineering and Knowledge Engineering*, Skokie, IL, 21-23 June 1990.

[5] Frakes, William, "Empirical Study of Representation Methods for Reuseable Software," *Software Engineering Guild Presentation*, Yorktown, NY, 7 February 92.

[6] Gagliano, R.A., M.D. Fraser, G.S. Owen, and P.A. Honkanen, "Issues in Reusable Ada Library Tools," *Emp. Found. of Info. and Software Science*, 1990, pp. 427-35.

[7] Goguen, Joseph A., "Parameterized Programming," *IEEE TSE*, Vol. SE-10, No. 5, September 1984, pp. 528-543.

[8] "IBM Reuse Methodology: Classification Standards for Reusable Components," *IBM Document Number Z325-0681*, 2 October 1992.

[9] "IBM Reuse Methodology: Qualification Standards for Reusable Components," *IBM Document Number Z325-0683*, 2 October 1992.

[10] Karlsson, Even-Andre, Sivert Sorumgard, and Eirik Tryggeseth. "Classification of Object-Oriented Components for Reuse," *Proc. TOOLS'7*, Dortmund, Prentice-Hall, 1992.

[11] Laitinen, Kari, "Document Classification for Software Quality Systems," *ACM SEN*, Vol. 17, No. 4, October 1992, pp. 32-9.

[12] Litvintchouk, Steven D. and Allen S. Matsumoto, "Design of Ada Systems Yielding Reusable Components: An Approach Using Structured Algebraic Specification," *IEEE TSE*, Vol. SE-10, No. 5, September 1984, pp. 544-551.

[13] Maarek, Yoelle S., Daniel M. Berry, and Gail E. Kaiser, "An Information Retrieval Approach for Automatically Constructing Software Libraries," *IEEE TSE*, Vol. 17, No. 8, August 1991, pp. 800-813.

[14] Pfleeger, S.L. Fitzgerald, J.C., Jr., " Software metrics tool kit: support for selection, collection and analysis," *Info. and Software Tech.*, Vol. 33, No. 7 September 1991 pp. 477-82.

[15] Prieto-Diaz, Ruben and Peter Freeman, "Classifying Software for Reusability," *IEEE Software*, January 1987, pp. 6-16.

[16] Prieto-Diaz, Ruben, "Implementing Faceted Classification for Software Reuse," *Comm. ACM*,, Vol. 34, No. 5, May 1991, pp. 88-97.

[17] Ostertag, Eduardo, James Hendler, Ruben Prieto-Diaz, and Christine Braun, "Computing Similarity in a Reuse Library System: An AI-Based Approach," *ACM Trans. on Software Engineering and Methodology*, Vol. 1, No. 3, July 1992, pp. 205-228.

[18] RIG Technical Committee on Asset Exchange Interfaces, "A Basic Interoperability Data Model for Reuse Libraries (BIDM)," *Reuse Interoperability Group (RIG) Proposed Standard RPS-0001*, 1 April 1993.

[19] Straub, Pablo A. and Eduardo J. Ostertag, "EDF: A Formalism for Describing and Reusing Software Experience," *Proc. 1991 Int. Symp. on Software Reliability Engineering*, Austin, TX, 17-18 May 1991, pp. 106-13.

[20] Yglesias, Kathryn P., "Limitations of Certification Standards in Achieving Successful Parts Retrieval," *Proc. 5th Int. Workshop on Software Reuse*, Palo Alto, California, 26-29 October 1992.

8.0 Biography

Jeffrey S. Poulin (poulinj@vnet.ibm.com) joined IBM's Reuse Technology Support Center, Poughkeepsie, New York, in 1991. His responsibilities on the RTSC included developing and applying corporate standards for reusable component classification, certification, and measurements. He currently works in Open Systems Development for IBM's Federal Systems Company and participates in the IBM Corporate Reuse Council, the Association for Computing Machinery, and the IEEE Computer Society. A Hertz Foundation Fellow, Dr. Poulin earned his Bachelors degree at the United States Military Academy at West Point, New York, and his Masters and Ph.D. degrees at Rensselaer Polytechnic Institute in Troy, New York.

Kathryn P. Yglesias (yglesias@vnet.ibm.com) is an advisory systems analyst in the IBM Reuse Technology Support Center where her work includes information model definition, classification evolution, and requirements definition for corporate standards and tools. She coordinated the IBM definition of formal methods for reusing non-code work products, especially customer documentation. Her experiences include engineering and project management for the Space Shuttle program and customer liaison for an internal computer systems organization. She is a member of the AIAA and Society for Software Quality.