Management of Reuse Libraries

Jacob L. Cybulski
Department of Information Systems
University of Melbourne
j.cybulski@dis.unimelb.edu.au

Abstract

We believe that any reuse-based software development process will utilise a large collection of reusable software work-products and components. Such a collection must be supported with a library system that could effectively and efficiently facilitate storage, classification, search and retrieval of reusable artefacts. The paper briefly reviews techniques that are and can be used in the organisation and management of a library of reusable software artefacts, preparation of these artefacts, their storage and retrieval. The section emphasises the need for artefact classification in support of effective library organisation. It compares and contrasts three main approaches to software classification, i.e. those based on keywords, facets and object-oriented techniques. It then discusses technologies that can support any of these three approaches.

1. Introduction

Management of reuse libraries, including artefact classification, storage, search and retrieval, is one of the most fundamental services expected of any reuse environment [2]. However, as there is a plethora of different artefact types of varying form and contents (Cf. Table 1) [99, p 752], the task of designing and the subsequent managing of even the simplest of the reuse libraries is non-trivial. The use of standard database or dictionary technologies can be applied with great success to the representation and holding of highly regular and structured artefacts, e.g. diagrams, tables or screen designs. Database and dictionary technologies, however, are not entirely suitable to handling ill structured and informal artefacts, in particular those based on plain English text (e.g. requirements, reports, schedules). Artefacts that lack structure and formality are hard to represent in an efficient form, hence, they require special-purpose processing and custom storage facilities.

According to a DoD / SRI report on Software Reuse [42], the most important functions and attributes of a reuse library are almost entirely determined by few carefully selected repository options, such as:

1. *Representation platform* that may include paper based systems, database management systems, information storage and retrieval systems, knowledge-based systems and hypertext.
2. *Indexing and classification methods* of which the most commonly used in practice are those borrowed from library science, i.e.
   - Free text keyword classification in which software artefacts are indexed by keywords extracted automatically from the text of the artefacts themselves;
   - Faceted classification where artefacts are categorised by synthesising vectors of facet values; and,
   - Enumerated or object-oriented classification in which reusable artefacts are assigned to mutually exclusive, hierarchical classes.

Enterprise:	**Application design:**	**Validations and verification:**
o Organisational structure	o Methodology rules	o Test plan
o Business area analyses	o Graphical representations	o Test data cases
o Business functions	o System diagrams	o Regression test scripts
o Business rules	o Naming standards	o Test results
o Process models (scenarios)	o Referential integrity rules	o Statistical analyses
o Information architecture	o Data structures	o Software quality metrics
	o Process definition	
Project management:	o Class definition	**System documentation:**
o Project plans	o Menu trees	o Requirements documents
o Work breakdown structure	o Performance criteria	o External/internal designs
o Estimates	o Timing constraints	o User manuals
o Schedules	o Screen definitions	
o Resource loading	o Report definitions	**Construction:**
o Problem reports	o Logic definitions	o Source code
o Change requests	o Behavioural logic	o Object code
o Status reports	o Algorithms	o System build instructions
o Audit information	o Transformation rules	o Binary images
		o Configuration dependencies
		o Change information

Table 1: Types of repository information[1]

Recently conducted experiments [50] showed no clear advantage of one method over another in terms of their recall effectiveness and precision of artefact search. The experiments, however, found that enumerated classification outperformed the other two methods in terms of speed of search. It was also determined that free text keyword classification is the least expensive as it does not require human intervention in the process of document indexing.

Other important methods of indexing reusable components are those drawn from:

- Artificial Intelligence, in particular computational linguistics, knowledge-based systems and expert systems;
- Hypertext; and,
- Formal specification methods.

3. *Repository scalability* across platforms of different size and complexity, which is of the highest importance to corporate reuse policies of the majority of large developer companies, such as Department of Defence or NASA. The issues considered as part of reuse scalability are library interoperability, library interface design, distributed heterogeneous databases, database security, quality assurance, change management, automated support for controlled vocabulary indexing; and better representation of library collections to help users find and understand the parts they need.

The following sections review examples of systems, methods and techniques that in recent times were adopted by research and development groups to support the construction and management

[1] Adopted from [99, p 752].

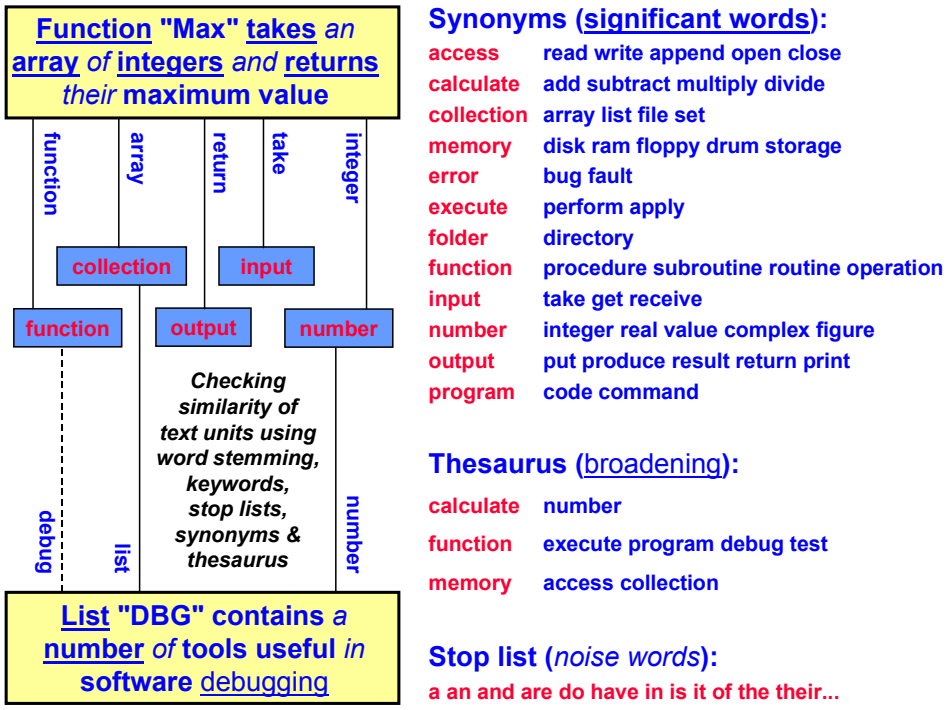

Figure 1: Example of keyword-based classification

of reuse libraries, and emphasise the approaches taken to the indexing and classification of reusable artefacts. Other factors, such as the repository platform and the scalability of the methods are also taken into consideration.

2. Keywords-Based Methods

Early software libraries provided very little automation to access and retrieve their functions, modules and data structures. Typically, e.g. in mathematical software libraries like IMSL [69], library providers delivered massive documentation with detailed description of each reusable component, a table of contents and index of names. In later years, e.g. UNIX or Microsoft Foundation Classes, the documentation was accessible in a form of searchable on-line manuals. It is only in the recent years, that libraries incorporated sophisticated classification methods based on the process of identifying and clustering keyword strings found in the text of the software artefact itself (Cf. Figure 1). Salton provides a thorough survey of keyword-based techniques as applied to text retrieval [111]. In his review he suggests the use of:

- *inverted indexes* for better access to text records;
- *distance constraints* to more accurately assess the nearness of two records;
- *weights and frequencies* to distinguish the importance of keywords;
- *stop lists* to eliminate commonly used and unimportant or noise words;
- *synonym lists and thesauri* to broaden the text retrieval queries;
- *word stemming and term truncation* to standardise word usage;
- *quorum-level searches* to control the size of retrieval output;
- *partial list searching* techniques to consider subsets of query terms;
- *phrase-formation techniques* to control keywords co-occurrence context;

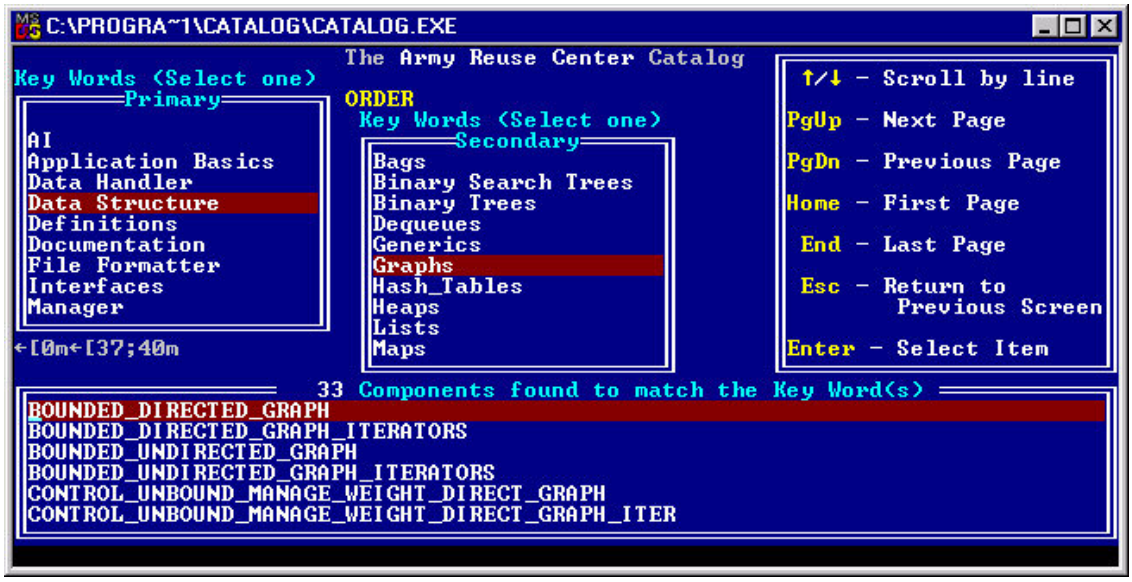

Figure 2: CATALOG system

- *statistical text indexing* to automate document classification;
- *document clustering* to improve retrieval of related documents;
- in Salton's later work [112], he also investigates various *linguistic approaches to document indexing*, e.g. generation of complex content identifiers, use of semantic terms obtained from machine-readable dictionaries, and the utilisation of specially constructed knowledge bases. He eventually highlights the unacceptable imprecision of the syntactic methods used in his experiments and points out the power of simpler statistical methods.

Nearly all of the techniques applicable to text retrieval have been tried and used with varying success in the classification and retrieval of textual software artefacts. Many of the identified methods have also been extended or combined with other types of software classification methods, i.e. faceted or enumerative (see the following sections).

2.1 Reuse with General-Purpose Document Libraries

As the majority of software artefacts are plain text, the most obvious approach to the classification and retrieval of software is to adopt an existing text processing system. Frakes and Nejmeh employed AT&T's CATALOG keyword-based information retrieval (IR) system to create a small library of software modules [49].

The CATALOG system (Cf. Figure 2) features a database generator, an interactive tool for creating, modifying, adding, and deleting records, and a search interface with menu and command modes. The search interface allows boolean combinations of terms and sets of retrieved records, and the queries are resolved with partial term-matching techniques, such as term stemming and phonetic variants. Searching is carried out using an inverted index of significant terms and a stop list is used to strip numerous noise words.

To further promote the clarity and accuracy of reusable information, Frakes and Nejmeh structured all of the reuse information using predefined templates to handle different types of software artefacts, e.g. modules and functions. The approach taken in this experimental system showed that standard information retrieval (IR) technology could be used effectively to organise simple reuse libraries. It also showed that organisation of large-scale repositories of software

documents needs specialised technology going far beyond simple IR. Such technology could assist in the interpretation, indexing and structuring of software artefacts across the entire software life cycle and it may call upon sophisticated information processing techniques, such as natural language processing, knowledge representation, production rules or the use of intelligent thesauri.

2.2 Reuse with Custom-Made Software Libraries

A specialised software reuse system best known for its keyword-based retrieval is DoD's SRL - STARS (Software Technology for Adaptable, Reliable Systems) Reuse Library. Experience with the construction and the subsequent use of SRL lead its developers to address many problems that in the past hampered efforts to effectively adopt standard text-retrieval techniques to software-reuse [9], e.g.

- stemming sometimes reduces unrelated words to the same stem;
- some apparent noise words are in fact contents words in some contexts;
- automatic use of synonyms, sometimes broadens the query in inappropriate way;
- certain combinations of words should be treated as phrases rather than search words;
- broadening of the queries with no hits is not always useful.

In view of these problems, SRL provided the following facilities:

- more accurate stemming based on the intended use of words;
- more *control* over the use of noise words, synonyms, and query broadening;
- *phrase matching* used in query refinement;
- help offered in the process of *query reformulation*;
- provisions in *handling technical expressions*.

These facilities lead to the following features incorporated in SRL:

- 15,000 word dynamic lexicon of citation word forms;
- chart parser built on top of repository services;
- graphical user-interface for reformulating queries; and finally,
- morphological analyser for handling technical terms.

2.3 Menu-Driven Library Access

The REUSE (REUsing Software Efficiently) system was built to effectively classify and retrieve existing software information [10], i.e. templates, modules, packages and executable programs. Similarly to SRL, the REUSE system uses keywords to classify its library components. At the same time, however, user access to the REUSE information retrieval system is organised not by means of queries, but instead, via a customisable, menu-driven front-end. The software uses keywords, which reflect the needs of an organisation, to build a *hierarchical system of menus* that reflect the organisational standards and methodologies. Such menus and keywords provide tools for classifying and retrieval of reusable software components. The REUSE system also maintains a thesaurus to reduce terminology differences within the user community.

2.4 Other Features

More recent, albeit less known, systems extend the fundamental keywords retrieval by including facilities that enhance the access and maintenance of reuse repositories.

In Ithaca's SIB (Software Information Base) library system, keywords characterising reusable artefacts are organised into *keyword descriptors*. These are subsequently weighted to determine their relative importance and then related by their semantic similarity. Both descriptors and their relationships form an elaborate network of nodes and links that could be later queried by the system search facility [52].

In the CART system (Computer-Aided Reuse Tool) classification is performed using keywords generated automatically from the specification models. Users can subsequently query the system using a much richer language than that used in the model. This is achieved with the assistance of a thesaurus, normalising the terminology, and some help from the user whenever the query resolution is too complicated for the system to undertake the correct decision [80].

In CodeFinder, the keyword-based retrieval system is further supported by iterative query reformulation and a novel retrieval approach using a neural-network like spreading activation and relaxation algorithm capable of effective retrieval of software components related to the query by keywords, phrases and lexical affinities [67].

3. Faceted Classification Methods

Colon classification and chain indexing, later known as the faceted classification scheme, was first proposed by an Indian scholar and a librarian, Shiyali Ramamrita Ranghanathan, as an effective technique for the management of library information [105]. Faceted classification is the main competitor to the popular Dewey decimal system of classifying library collections. Prieto-Diaz and Freeman were first to suggest the possibility of adopting faceted classification for the classification and retrieval of reusable software artefacts [100, 102]. Traditionally, the faceted classification technique relies on the existence of a large number of domain terms organised into several distinct and orthogonal sets referred to as *facets* (Cf. Figure 1). Each newly acquired artefact is subsequently described in terms of a *descriptor vector* where each vector value is picked from one of these pre-defined facets. Such classification vectors are then stored in a relational table in which columns represent facets and rows denote artefact descriptors. The classification table can later be searched to find and retrieve the necessary artefact descriptors. Searching for matching artefacts could be simple and effective with the use of a relational query language, such as SQL. An alternative (and preferred) method of artefact retrieval is based on a metric assessing the *conceptual closeness* of a query, artefacts and their attributes in each of the facets. Other search and matching mechanisms may utilise more sophisticated methods based on the statistical profiles of artefact descriptors, affinity and similarity metrics, vector spaces or fuzzy logic [102, 111].

When it comes to the speed of retrieval and the cognitive complexity associated with the use of the artefact indexing and retrieval mechanisms, faceted classification compares well with other classification schemes, e.g. keyword-based, attribute-value and enumerative [50]. Its main strength is ease of artefact classification, simplicity of representation and storage of artefact descriptors, uniformity of classification attributes, and ease of automation [101]. The main deficiency of faceted classification, as adopted in many existing systems, is the high cost of repository maintenance, which is due to the predominantly manual classification of artefacts [89, 118].

The following aspects are characteristic for many faceted classification systems:

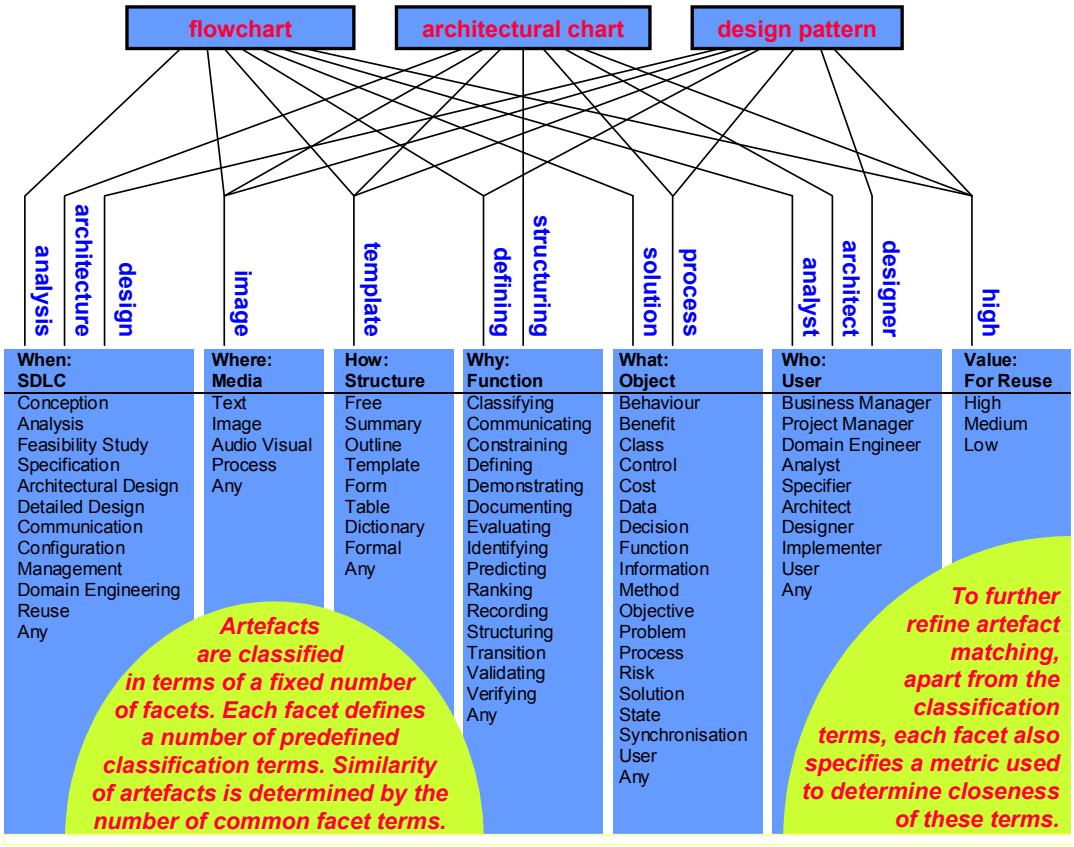

Figure 1: Example of faceted classification

- facets specify a *controlled vocabulary* used in the artefact classification;
- facets are usually *tailored* to make the classification subject-specific;
- classification terms are commonly *broadened* with the use of synonym lists and a thesaurus;
- facets are *ordered by their relevance* to the users of the collection;
- similarity of facet terms is assessed by arranging them into a *conceptual distance graph*;
- retrieved items are *ranked* on their semantic closeness to the query;[2]
- faceted classification is *extendible* as new classification terms and new facets can be added at any time;

3.1 Artefact Management with Facets

In the scheme proposed by Prieto-Diaz and Freeman and adopted by GTE Data Services in their Asset Management Program [103], software components are described in terms of the following facets :-

- function,

[2] Which is an artithmatic combination of semantic distances between the facet terms characteristic of the query and the retrieved artefact.

- objects the function manipulates,
- medium supporting the object structures,
- the system type,
- functional area, and
- application settings.

Classifying the component, hence, consists of selecting a tuple that best describes the component. To avoid multiple classifications due to word synonyms, a controlled vocabulary is provided via a *thesaurus*. In addition to the list of characterising term, each facet also defines a *weighted conceptual graph* of its classification terms. The graph is then used to calculate the conceptual closeness between terms and the similarity of classified artefacts and the query used to retrieve them. Retrieved programs are then *ranked*, using fuzzy set theory, according to the estimated reusability effort based on:

- program size in lines of code,
- program complexity in terms of the number of modules, linkages and cyclomatic complexity,
- program documentation quality,
- subjective ranking,
- programming language in relative language closeness, and finally
- the reuser's experience.

3.2 Combining Search Techniques

Sorumgard, Sindre and Stokke [118] also apply faceted classification as the main information retrieval mechanism in their ESPRIT project REBOOT. They observe that in faceted classification reuse decisions are based on information coming from several different categories as opposed to a single source of reuse decisions coming from a single classification hierarchy in enumerative methods. From this viewpoint, faceted classification is superior to commonly used enumerative or taxonomic schemes.

The REBOOT search for reusable components invokes either an *associative search* (on attribute value, e.g. DBMS) or *textual search* (pattern matching or linguistic attributes), both types based on the limited dictionary of classification terms. Initially REBOOT used four facets:

- abstraction (e.g. stack, resource manager),
- operations (e.g. push, pop),
- operates on (e.g. integer, set, list, resource), and
- dependencies (e.g. compilers, operating systems)

Through the application of their scheme, the authors found a number of problems in maintaining the term-space to keep the classification and the term-space consistent. They also found some problems with the granularity of the term-space. After a thorough evaluation of their experiments they proposed to alter their classification scheme to include the abstraction, operations, parts, collaborators, and dependencies facets. For each facet they identify its *main classification terms* and the additional *qualifying classification terms*.

3.3 Document Clustering and Concept Affinity

Although, the F3 (From Fuzzy to Formal) project does not explicitly list faceted classification at the center of its retrieval methods, Castano and DeAntonellis use a hybrid approach using the techniques of faceted classification, document clustering, and semantic affinities [26]. In the F3 reuse system, components are extracted from previously constructed applications and then they are grouped on the basis of affinity coefficients. This group of similar application components, or schema, defines a single reusable component abstracting structural and behavioural features common to all of the components belonging to that group. The system assumes a specific domain of application for which it provides a thesaurus of synonyms and homonyms for the frequently used terms (to be used later by the reuse similarity mechanism). As F3 addresses the entire reuse cycle, Castano and De Antonelli specify the details of reuse techniques to be used in the design-for-reuse and design-by-reuse. Design-for-reuse involves:

- selection of schemas based on the *schema descriptors*;
- classification of candidate schemas by grouping them into *similarity clusters* based on a single link clustering method [111, pp 329-333];
- *semantic affinity levels* based on a complete-link clustering method [111, pp 334-336] are used in the selection and classification of candidate components;
- design of reusable components;
- assimilation of reusable components into a library.

Design-by-reuse consists of the following phases:

- retrieval of reusable components by formulating a suitable query involving component descriptors;
- schema design; and its
- transformation by refinement and abstraction.

3.4 Other Features

Embley and Woodfield combine faceted indexing and keyword techniques to classify a collection of abstract data types [44]. Each abstract data type had associated descriptors that classify it into a number of facets consisting of descriptive keywords and their aliases. Descriptors can also specify relationships between pairs of abstract data types, some of which can be generated automatically from the keyword values used in the descriptors. In ESF ROSE [93] faceted classification is also used to classify software components. Apart of the standard features of limited vocabulary, use of synonyms and thesauri, the system also has a unique ability to use facets structured into several different levels of abstractions. This levelling of facet information enhances the effectiveness of artefact classification but also aids the retrieval processes and clustering of artefacts into similarity groups.

It should be noted that faceted classification finds its way into many other computer-based applications, starting with the original library applications [98] and ending with organising database access [43].

4. Object-Oriented Classification Methods

A drastically different way of organising software artefacts can be achieved with the use of enumerative classification [100], which relies on entering artefacts into a predefined hierarchy of categories. One of the most successful application of this method is Dewey decimal method of classifying book titles by their subject [e.g. see 37] (cf. Figure 1). Another form of such hierarchical classification is represented by object-oriented systems of concepts [41, 58, 61].

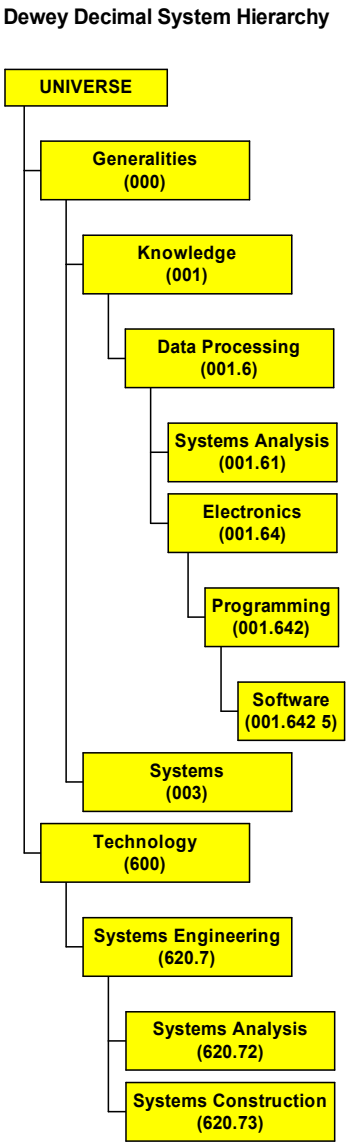

Figure 1: Example of enumerative classification

Object-oriented systems typically consist of many inter-related concepts, commonly referred to as objects (also known as nodes, concepts or instances). Objects are described in terms of their property values that may include references to other objects. Objects can also define methods (also known as member functions) producing useful computations in response to requests received from other objects in the system or from the system interfaces. New objects with all their properties and methods are created (instantiated) according to abstract descriptions provided by classes (frames, schemata or units). Classes are organised into inheritance (subsumption) hierarchies (taxonomies). The more general classes are said to subsume the more specific ones. As a result, the more specific classes inherit the properties and behaviour of their superclasses. Subsumption also implies set-superset relation of the associated class instances.

Different classification approaches may have to be taken in relationship to different types of objects in the software system. We may wish to build a separate taxonomy for class specifications and designs, design alternatives, architectural peculiarities or the details of class implementation [35].

Advocates of object-oriented methods [87] claim that the structures and mechanisms employed in object-oriented software systems greatly facilitate reuse of software components:

- Object-oriented programming promotes bottom-up development using existing classes;
- An object-oriented system can be viewed as a collection of related and communicating classes rather than a monolithic block of program statements;
- Classes combine data and procedures, they are modular and general, their interfaces are clearly specified and abstract;
- New classes can be derived from the exiting classes by specialisation and instantiation;

- Features and behaviour of existing classes are readily available in the newly derived classes via inheritance and multi-inheritance;
- It is possible to develop classes specifically for reuse, i.e. they may include a number of deferred features (e.g. virtual methods), which are then implemented at the class specialisation;
- Classes combine elements of design and implementation into a single reusable unit.

In the past, there has been a tendency to equate reuse with object technology or to assume that reuse cannot occur without it. Field studies show, however, that highly acclaimed object-oriented programming languages, such as Eiffel, Smalltalk, C++ and Java, Ada 95, Object Cobol and Object Pascal, do not provide any significant reuse benefit on their own [63].[3] In fact to date, the most successful large-scale reuse has been achieved with Cobol, Fortran, C and Ada [106].[4] Object-oriented tools support artefact abstraction and classification, encapsulation and modularity, inheritance and polymorphism, aggregation and composition, concurrency and persistence. Still, the spectrum of reuse tasks supported by such tools is usually limited only to the creation and composition of reusable artefacts - hence it is very narrow. Object-oriented methodologies and tools supporting them must still be created to take advantage of the potential reuse features provided by these languages.

It seems that objects are neither necessary nor sufficient for effective reuse. They could, however, prove extremely important if the right approach to their life-cycle utilisation is employed [62].

4.1 Artefact Relationships in Semantic Networks

The earliest systems of structured object descriptions were semantic networks [104] - a collection of concepts and their property-values, all forming a network of nodes inter-related by their properties. Although such systems were not designed to be repositories of software components, many techniques developed primarily for the manipulation of knowledge in semantic networks are still applicable to the processing of software artefacts and their descriptions.

Semantic networks are structured into planes of nodes related to a single concept. Individual planes, however, have no internal structure, hence, a single semantic concept is represented as a web of connections to all possible reachable nodes in that plane. A concept may also associate with concepts from other planes; such links represent inferences and shifts of attention. In the semantic network there is no predetermined hierarchy of classes and superclasses; every node defines its own hierarchy of concepts reachable from its point in the network.

Search and matching of the semantic network concepts may rely on the idea of activation spreading with decay. In this method, a number of nodes representing the query are marked as active. This activity will then spread to other areas of the network. The activation will decay with distance from the starting points, the closer the higher activation, the further the weake. Activation of nodes is additive, hence, the concepts receiving activation from many sources will

---

[3] Programming languages provide only a method of defining structured objects, they do not supply any methods of organising, searching and managing repositories of such objects.

[4] Such successes are mainly due to the highly regulated industries using these languages and the nature of projects undertaken in these languages, i.e. military, engineering and space-program applications.

be highly active, these concepts are considered to represent the answer to the query. While the activation spreading within a single plane represents a mere recall of associated information, plane "hopping" is regarded as inferring ideas not directly represented in the data given in the concepts being matched. To enhance processing of information in semantic networks, some systems allow linking of actions and inferences to the nodes or associations between such nodes. Such actions and inferences are triggered during various operations on the knowledge encoded in the network, e.g. search, matching, addition of new knowledge, detection of redundancies, conflicts, etc.

Semantic network models of human memory include HAM [7] and ACT [6]. The best known examples of useful applications of knowledge representation schemes derived from semantic networks include vision and scene recognition systems [123], natural language processing [85, 113, 122] or databases and expert systems [119].

4.2 Component Taxonomy with Subsumption

Frame systems extended the notion of semantic networks, thus allowing knowledge to be structured into a collection of frames (or classes) - reusable abstract knowledge units [91]. Frame properties are called slots (properties or variables) and they may store both declarative (values) and procedural (methods or demons) knowledge. Frame systems are usually organised into PART-OF and IS-A hierarchies to allow composition and inheritance of properties shared between general and specialised frames. Frame systems are a direct precursor, and are isomorphic, to object-oriented systems. Hence any technique developed for frame based systems will also be applicable to object-oriented systems.

Classification in frame or object-oriented systems is the operation of assimilating a new description into a taxonomy of existing concepts by linking it to its most specific subsumers and the most general concepts that the description subsumes [126]. Woods lists five types of subsumption relations:

1) *Extensional subsumption* where the model-theoretic extension of a concept contains the extension of another concept;
2) *Structural subsumption* where the concept subsumption is determined by virtue of formally specified criteria applied to the structure of the concepts;
3) *Recorded subsumption* where general concepts are explicitly recorded as subsuming the more specific concepts in a stored taxonomic structure;
4) *Axiomatic subsumption* where the more general concept is asserted to subsume the more specific concept as an axiom of the knowledge base; and
5) *Deduced subsumption* where the more general concept is deduced to subsume the more specific concept by deductive inference applied to knowledge of the domain.

The majority of existing object-oriented systems employ extensional or recorded subsumption leading to a manual process of classification and retrieval of objects by the designers of object systems. Classification tasks, however, can easily be automated when the system relies on structural, axiomatic or deduced subsumption.

In general, the process of classification of information and its subsequent search in the class taxonomy relies on the traversal of the subsumption hierarchy from the most general to the most specific classes, looking for successful matches against the query [75]. The query is usually defined as an object partially filled with desired property values or constraints imposed on these properties. The non-matching classes and all of their subclasses are cut from the search-tree.

Searching the subsumption sub-trees for the more specific candidates further refines the list of matches. In the simplest case the search can be breadth-first or depth-first. Other search heuristics may also be used to improve the chance of a successful match [116, cf. Search]. Classification of new objects and their instantiation into a subsumption taxonomy of existing classes uses a very similar process. A query object is constructed from the observed feature values and a standard search algorithm is used to determine a set of matching candidate superclasses and subclasses. Depending on the quality and the number of matches, the instantiation heuristics may determine to instantiate the new object into one of the found candidates, or a new sub- or super-class may need to be defined, or existing classes may be split or joined. Early frame representation languages and systems include FRL [107], KRL [17], NUDGE [60], PERL [38] or KEE [46].

A typical frame or object-oriented system derives its power and efficiency from its highly modular representation, inter-linked class hierarchy and the subsumption-based inheritance mechanism. KL-ONE [19] and its descendants, such as LOOM, KRYPTON, KANDOR, BACK and CLASSIC, however, have been characterised as "frame systems implemented in logic". Therefore, KL-ONE languages have a number of architectural features that make them unique amongst other subsumption-based systems [83]:

- They are *logic-based* (first-order logic);
- They draw the distinction between a *terminological* and *assertional* knowledge to express concept descriptions and to state facts about knowledge domains respectively;
- They include a *classifier* that organises concepts into subsumption taxonomy.

Using logic as the mechanism behind KL-ONE inheritance offers the system users a number of benefits:

- users can inspect individual concepts to see if the constraints that logically apply match the user's own expectations;
- the system automatically detects concepts that inherit conflicting constraints;
- inherited constraints and class-superclass and instance-class subsumption links are cached to further improve the retrieval performance.

Computation of subsumption relationships is usually quite inefficient, and in general, even untractable. KL-ONE languages, however, choose only a subset of inferences useful in determining instance-class and class-class relationships. The selected subsumption inferences eventually lead to good performance indicators in CLASSIC, the final attempt at automating classification of logic descriptions. The main deductive and classification power of CLASSIC [18] comes from its rich collection of inference rules, which govern the completion of logical consequences and assertions, classification and subsumption of individuals and classes, and application of forward-chaining rules. CLASSIC also specifies a series of methodological steps (some are machine-assisted) that enhance the system's knowledge-engineering tasks, i.e.

- enumerating object types,
- distinguishing concepts from roles,
- developing concept taxonomy,
- determining value restrictions,
- detailing unprecedented value restrictions,
- determining inter-role relationships,

- distinguishing essential and incidental properties,
- distinguishing between primitive and defined concepts, and finally
- determining clusters of concepts.

Taxonomical organisation of software artefacts is very popular in those systems that store and reuse object-oriented software components, many of which have already been partially classified in the process of their design and implementation. Devanbu, Brachman, Selfridge and Ballard [40] describe LaSSIE software information system that integrates architectural, conceptual and code views of a large software system. The system query and browsing facilities allow programmers to search a large software system in an attempt to identify reusable software components suitable to the task at hand, hence, overcoming what the authors term *the discovery problem*. LaSSIE's knowledge base stores the software components in a taxonomy of frames that represent actions (e.g. external and internal actions, and stimuli), objects (e.g. communication device, resource, hardware and software), doers (e.g. users, processes, groups and processors), and states (e.g. line, resource, data and network states). LaSSIE's knowledge-based system, KANDOR derived from KL-ONE, represents and classifies software components in frames with logic. It allows:

- aggregation of information about programs and their components;
- semantic retrieval of software components;
- use of classification and inheritance to suport software updates; and
- use of a knowledge base as an intelligent index of software artefacts.

The retrieval algorithm used in LaSSIE takes a query frame, places it in the frame hierarchy and then matches all of the instances that a query frame subsumes. LaSSIE also allows the use of natural language queries that can be converted into query frames. In later years, LaSSIE's original representation language KANDOR was replaced with a newer knowledge-based system CLASSIC [18]. The system significantly extended the original functionality built into LaSSIE. It offered a very extensive set of tools to classify abstractions and their instances, to search for class and object descriptions, to check system integrity, and a great many facilities that were used to support many other knowledge engineering tasks - all being at the crux of LaSSIE's software reuse operation. While LaSSIE proved to be a very useful software retrieval tool, building its vast knowledge base is still a manual, hence slow and expensive, process.

The amount of manual effort expended in the description and classification of reusable artefacts can be reduced through the use of a knowledge acquisition environment tailored for a reuse-oriented model of the software development process. BAUHAUS [5], developed in the ART language, provides such an intelligent environment to Ada software libraries. The system automates several programming tasks found in a typical software development life cycle, i.e.:

- storing and maintaining information about program parts and their interconnection;
- composing specification of new applications from reusable parts; and to
- generating code from specifications into a given target language.

BAUHAUS assists programmers in all these activities by:

- initialising its knowledge base by parsing text specifications and source libraries into a frame-based description language;
- providing facilities to manually annotate acquired objects with requirements information;

- classifying the resulting component descriptions into a subsumption network - thus giving requirements, specification and implementation views of the reuse system.

Once Ada components have been classified into a subsumption structure, BAUHAUS provides component browsing and retrieval facilities based on the retrieval-by-reformulation approaches also used in knowledge-based systems.

The AIRS system is another example of a frame-based system to facilitate reuse of software components [97]. The system uses frames to represent the features and composition of program functions, as well as software components "subsumption" and "closeness" relationships, to facilitate searching for reusable operations and packages, and to provide capabilities for helping programmers to assess the worth of reusing particular packages.

4.3 Hybrid Systems

In some applications, the class hierarchy may be very large or it may include multi-inheritance relationships between classes, hence, leading to lattices of classes. In such cases, other forms of class indexing and classification have been suggested to assist the search for matching classes. In the ITHACA project and more specifically in its RECAST specification reuse subsystem, the taxonomy of artefact descriptions can either be browsed along the structural and methodological links, or it can be queried using a combination of class static and dynamic attributes, keywords with thesaurus, and weights [13, 52]. DEROS system of object-oriented components is built upon a hypertext system and faceted classification of domain knowledge and reusable components' information to improve access to individual artefacts [27]. In WharfRat [84], a semantic network of data types is used together with fuzzy logic as its primary artefact retrieval mechanism.

4.4 Large Components

In the majority of circumstances individual classes cannot be considered useful reusable units - they are too small and their interfaces are too specific. Some authors propose the reuse of larger and more abstract groups of objects, such as object architectures, design and analysis patterns, abstract subsystems and frameworks [22, 48, 53, 71, 125]. Some software engineering researchers report having demonstrations of such object abstractions and macro-structures to have a very significant impact on software reusability, e.g. see use of:

- view-controller-model framework in Smalltalk-80 [39];
- frameworks in event-based simulation [90];
- domains and themes in CASE reuse [64];
- use-cases and architectures in domain analysis [28].

Others contest this claim [21], claiming that even such large-grain software components do not directly contribute to improved reusability due to the lack of relevant methods and techniques that could assist software engineers in their classification and storage in the reuse repository and the subsequent retrieval and adaptation.

5. Technologies Assisting Artefact Reusability

There exist techniques and methods drawn from many areas of computer science, that do not provide a direct functionality to reuse libraries but which are commonly used to enhance some of

their aspects, e.g. usability of a query language, effectiveness of artefact access or representation of artefact description. Technologies most useful in information retrieval are those provided by work in text and knowledge processing, e.g. techniques to assist in text scanning and classification, routing, reformatting, data extraction, selective retrieval, text parsing and representation, linking and referencing, query answering, etc. Engelien and Ronnie [45] demonstrate the feasibility of natural language and knowledge representation techniques in many commercial applications (though not related to software engineering), e.g.

- financial telex scanners - ATRANS and CBR/Text (Cognitive Systems Inc, pp 183-186), IBS (Citibank, pp 107),
- understanding sketchy messages - Nomad later called Vox (U.S. Navy, pp 109-110),
- recording ship movement (Cognitive Systems for U.S. Cost Guard, p 109),
- information retrieval for Yellow Pages (GSI-Earli, pp 190-194),
- financial news service - SCISOR (General Electric, p 113);
- news classification and retrieval - TIS and Textline (Reuter, pp 115, 266-269),
- patents classification - Realist (Siemens Nixdorf Informationsysteme for U.S. Patent Office, pp 118-119, 221-227),
- etc.

The techniques and methods developed in these systems also found their way to software reuse.

5.1 Hypertext

The modern notion of hypertext is defined as a collection of connected documents that can be traversed with the aid of computer either linearly or via hyper-links that allow rapid transfer between documents [23, 96]. Hyper-links are usually defined between related concepts in different documents, semantic associations, term uses and their definition, etc. Commercial hypertext systems are assisted with many other editing, annotation, browsing and searching tools, such as hypertext editors, databases, navigation diagrams and maps, backtracking and history mechanisms, search engines, etc. Current developments in World Wide Web technology promote many practical applications of hypertext to wide-area networking.

Applications of hypertext to software development environment range from finding and navigating between program components [31, 88], organising CASE repositories [16, 34], maintaining software life-cycle documents [54, 55], managing development and organisational knowledge [4, 24], supporting team design deliberations [30], dealing with requirements elicitation [73, 74], etc.

Although many reuse projects rely on the use of hypertext, they are not restricted to the simple hyperlink access to artefacts. Instead, they frequently combine hypertext function with query languages, hyper-maps, history-mechanism, commonly used with generic hypertext systems as well [4, 23, 25, 29].

5.2 Natural Language Retrieval

Syntactically and semantically driven natural language parsers found wide-spread applications in text retrieval systems, knowledge-based information management and also processing of program, specification and documentation texts. Hahn [65] argues that such an approach is of

particular value in large-scale text analysis that involves extraction of information from text, generation of informative text from DBs, summarisation of text, translation, monitoring and routing of texts, classification and retrieval, localisation of relevant portions of text relative to the request. Hahn provides a number of requirements of the NLP system for information retrieval (and hence, software information retrieval systems as well):

- grammars must provide a sufficiently broad coverage;
- analytic machinery must be robust with respect to the ill-formedness of language;
- grammar and parsers must be maintainable and extendible;
- grammar must handle macro structures of text;
- to minimise the computational effort grammar and domain specification must be adaptable or include heuristics;
- computational costs must be within the acceptable feasibility regions.

Many applications in general-purpose information retrieval adopted the principles outlined by Hahn and others [33, 78, 115, 117].

Natural language processing also found its way into supporting software development tasks, e.g. automatic programming [66] , program design [1, 14], development of executable specifications in natural language [51, 114], formalisation of informal specifications [11, 12, 15, 109, 120], paraphrasing formal specifications in natural language [110], requirements acquisition and specification [36, 94, 108], requirements validation [95], recording design reasoning [127], etc.

Few attempts have been made to apply natural language processing techniques to facilitate software reuse. The most prominent natural language assisted reuse projects include the following.

Girardi and Ibrahim [56, 57] describe their work on the reuse system that takes natural language descriptions of software components, uses lexical, syntactic and semantic information to classify them into a knowledge base of frames. The retrieval process is capable of converting a natural language query into a frame representation and then uses the similarity measure based on conceptual difference of terms contained in a thesaurus of term synonyms, generalisations and specialisations. Their classification system relies on the representation of text in terms of Fillmore's case structures [47]. The two phrases are considered to be similar when a conceptual distance between them is small. The distance between two phrases is defined as the distance between phrase heads plus the distances between their respective head modifiers. The distance between individual terms is defined in an a-priori fashion.

Conceptual distances, based on lexical affinities, help a requirement analyst find abstractions in problem descriptions in GURU and AbstFinder [59, 82]. Both environments use natural-language descriptions of software components to index them, retrieve them and navigate between them. The authors use uncontrolled vocabulary so that their approach could be easily scalable in the future. In their approach they calculate word affinities by counting the frequencies of pairs of open-class word roots that are separated one from another by no more than five words in the same sentence. In their analysis they rely on the word's information contents measures and the observed probability of word occurrence in the corpus of representative text. To compare relative performance of the resolving power of different affinities across all of the documents, they standardise it with respect to its average and standard deviation. To index software documents the

authors build an inverted file of affinity tuples. To browse a collection of system information, documents are clustered around similarity groups [3].

Natural language processing techniques have also been used by Naka to define, reuse, validate and transform software requirements [94]. In his system, specifications are entered in structured Japanese, then they are automatically formalised, analysed for reusability and verified, and finally transformed into Lisp programs. The system features Japanese natural language interface, information retrieval system, repository of components, and the system structuring components.

In other systems, software reuse may be additionally enhanced with the use of natural language query language [20], lexical analysis of terms, syntactic processing of text to detect keywords [112], etc. All such facilities are frequently used in combination with other retrieval methods.

5.3 Analogical Reasoning

The ability to reuse software is inherently linked with the ability to recognise, classify, learn, and reason about patterns found in applications, software components, their attributes and the processes leading to their creation. It is also the ability to perceive similarities and reason by analogy - one of the most fundamental aspects of human cognition [121]. The theory of analogical reasoning explains how people judge and track similarity of concepts, how they account for conceptual changes in development and the acquisition of knowledge, and how new concepts are formed based on the representation and structure of existing knowledge. The use of analogical reasoning can also be used to explain cross-domain reasoning, dealing with ill-structured problems, and restructuring of conceptual representations [68].

Winston [124] was one of the first to use the theory of analogical reasoning to implement a working computer system to acquire, classify and use knowledge of a given problem domain. In his system, analogy is used to answer questions about one situation, given another situation that is supposed to be a precedent. His system had a number of key features, e.g.

- the system was capable to dynamically add new facts to its knowledge base and to refine existing relationships as it discovered new facts by analogy;
- the system matched its knowledge representations based on the importance of their features, hence, paying less attention to unimportant concepts and their properties;
- it learned new facts given preference to the utilisation of analogical propagation of constraints and reasoning;
- the system exploited rich knowledge classification scheme to identify major representational features.

Winston determined that to effectively match analogous parts (using importance-dominated matching), he required utilisation of a broad spectrum of information about these objects, e.g. properties and relationships, corresponding comments, classification information, importance of information, macro-structures and abstractions, and a variety of information measures. Anderson and Thompson [8] further investigated the use of analogy to develop computer program able to learn new facts in a schematic knowledge representation system. They developed the PUPS system able to effectively discover missing object properties, functions and forms. They also developed sufficient theory to propose methods for analogical concept refinements, knowledge compilation, discrimination learning of new concepts, etc.

Maiden and Sutcliffe [86] proposed to use the mechanisms of analogical reasoning to guide the process of software reuse. Their primary motive for the approach was the observation that high-level artefacts, such as complex requirements specifications need very complex knowledge for their effective reuse. They, hence, suggested that simple reuse techniques, such as those based on keywords or facets, may be unsuitable to complex problem domains. Instead they decided to use sophisticated knowledge-based problem representation and the use of analogy as the principal mechanism for the retrieval of reusable software components. Their reuse system included the Analogy Engine as its core component. The engine was capable of matching specifications semantically, structurally, pragmatically and abstractly. It consisted of two subsystems, i.e. the Analogy Matcher that identified candidate analogical mappings with abstract domain models and the Abstraction Selector which reasoned heuristically about critical differences between candidate abstract domain to further refine their selection. Maiden and Sutcliffe's Analogy Engine used a simple predicate representation of domain concepts to encode object structures, state transitions, object types, conditions on state transition, transformations, external transition events, and information system requirements.

A similar system was also used by Lung and Urban [81] to classify program concepts, to bridge of the gap between generic domain abstractions and features in specific application domain and to determine general software attributes. However, in their work, the authors describe analogical classification, which is a combination of faceted and enumerative approaches.

Other attempts at analogy-based reuse include applications in the reuse of software designs, formal specifications [76, 92], etc.

6. Comparison of Methods

We have introduced several methods, techniques and tools useful in the organisation and management of reuse libraries. All of the introduced concepts were grouped into three major classes of library mechanisms based on the method of classifying reusable artefacts, i.e. keyword-based, faceted and enumerative (or object-oriented). We have also identified a number of technologies enhancing reusability of library components. Table 1 summarises and compares all of the discussed methods.

Comparing each of the groups of the reuse methods and techniques, we come to the conclusion that there is no clear advantage or disadvantage of their fundamental features. However, at the same time, a number of studies were conducted to determine the methods' usability and their effectiveness and efficiency of artefact retrieval. There seems to be a strong disagreement on the data obtained from various experimental studies, which clearly shows that each of the methods has its own strengths and weaknesses and can be used in some practical applications. In the majority of cases, the most effective systems are based on the combination of methods, either in the classification itself, the search or pre and post-processing of queries.

For example, Croft compared the statistical and knowledge-based approaches to information retrieval [32]. He believes that the two areas of research can complement each other rather than be in competition. He concedes that while both keyword-based and statistical methods are very efficient, their search mechanisms not easily tailored to the needs of individual user, nor do they facilitate making use of the structure and contents of the application domain. Knowledge-intensive and heuristic methods, however, can provide this additional functionality to the traditional information retrieval techniques. Jacobs further advances the idea of using hybrid approaches to information retrieval, in particular he deeply believes that statistical methods can significantly assist in the identification of the significant terms to be used in knowledge-intensive natural language applications [70].

Table 1: Comparison of methods useful in the library management

Method	Pre-Processing	Repository	Classification	Search
Keyword-based	lexical analysis word stemming phonetic variants stop lists phrase formation parsing thesaurus with synonyms	inverted index full-text database	statistical text indexing free vocabulary term clustering document clustering	distance constraints quorum-level searches partial list searches menus query reformulation
Faceted		relational database descriptor vectors tailored facets conceptual distance graphs	manual controlled vocabulary conceptual closeness fuzzy relevance	
Object-Oriented		single aspect subsumption hierarchy class-superclass attribute-value	specialisation generalisation instantiation	taxonomy traversal menus pattern-matching
Assisting Technologies	Natural language processing Hypertext Analogical reasoning			

A number of people compared the effectiveness of queries in natural language vs. queries constructed in a formal language, e.g. SQL [72]. In Jarke *et al* experiments, SQL performed better than natural language on a variety of measures, but natural language queries required less effort to use. Natural language queries were found to be more concise and required less formulation time than those expressed in SQL. Their experiment also shows the importance of feedback, iterative refinement of queries, training in the use of restricted natural language, and other factors related to the total operating environment on the overall performance of users. A performance evaluation of 15 text-analysis systems was recently conducted to realistically assess the state of the art for detailed information extraction from unconstrained continuous text [77]. The competing systems were evaluated for recall, precision, and over-generation. Their experimental results show that systems incorporating natural language-processing techniques are more effective than systems based on stochastic techniques alone. Quite a different result was obtained by Lewis and Spark-Jones [79] who clearly distinguish between text retrieval (TR),

document retrieval (DR) and knowledge retrieval (KR). They show that some statistical methods are extremely useful, especially that in recent years it is hard to draw a clear distinction between these methods and those using natural language processing techniques. As they point out many of the stochastic techniques actively utilise the low-level NLU techniques as well.

Best known experiments in the use of natural language processing techniques in information retrieval have been conducted by Salton. They showed much inefficiency and inaccuracy associated with the use of such natural and vague user interface. He found that these problems can be easily bypassed with the use of statistical methods [112]. At the same time, other studies [117] compared natural language systems of different lexicon and grammar structure and size against stochastic systems. When the systems were ranked according to the highest combined recall and precision scores, the top 8 systems were all natural language processing systems. Analysis of the top systems showed that:

- text analysis techniques progressed far beyond database interface applications;
- natural language systems outperformed the traditional IR systems;
- available techniques for semantic and syntactic processing are sufficient in text analysis, except for discourse analysis;
- throughput exceeded 5 times that of human encoders, but the precision and recall were not as good;
- an average of 1 man-years were put into development of these systems;
- the top scoring systems incorporated a diverse range of natural language techniques.

Frakes and Pole conducted an experiment, which is most relevant to our research, to compare different methods of software component retrieval based on attribute-value, enumerated, faceted and keyword-based classification [50]. They found that there are no significant differences between four methods in terms of their effectiveness, as measured by recall and precision. Different methods found different, though similar, items. Users had no clear preference for a representation method, however, there were significant differences in user-search times between all methods. In a more recent experiment, Mili, Ah-Ki, Godin and Mcheick [89] disputed the results obtained by Frakes and Pole, claiming that free-text retrieval is far more superior to faceted and keyword-based classification and retrieval. However, their experiments covered very few subjects and are based on few queries.

7. Summary

Based on the results of experiments, we believe that all approaches to repository organisation and its access are similar. Looking at the features of each method, it seems that faceted classification has many advantages over other methods, e.g. its storage facility is compatible with standard relational database system, its classification is simple, retrieval based on the conceptual closeness of artefacts is intuitive. The main disadvantage of the method is the high cost of its manual classification. Should a technique be found to improve this process, its many advantage will outweigh its disadvantages.

8. Bibliography

1. Abbott, R.J. (1983): *Program design by informal English descriptions.* Communications of the ACM. **26**(11): p. 882-894.
2. Agresti, W.W. and F.E. McGarry (1988): *The Minnowbrook Workshop on Software Reuse: A summary report*, in *Software Reuse: Emerging Technology*, W. Tracz, Editor. Computer Society Press: Washington, D.C. p. 33-40.
3. Aguilera, C. and D.M. Berry (1990): *The use of a repeated phrase finder in requirements extraction.* Journal of Systems and Software. **13**(3): p. 209-230.
4. Akscyn, R.M., D.L. McCracken, and E.A. Yoder (1988): *KMS: A distributed hypermedia system for managing knowledge in organisations.* Communications of the ACM. **31**(7): p. 820 -835.
5. Allen, B.P. and S.D. Lee (1989): *A knowledge-based environment for the development of software parts composition systems*. in *11th International Conference on Software Engineering*. Pittsburgh, Pennsylvania: IEEE Computer Society Press, p. 104-112.
6. Anderson, J.R. (1976): *Language, Memory and Thought*. Hillsdale, N.J.: Lawrence Erlbaum.
7. Anderson, J.R. and G.H. Bower (1973): *Human Associative Memory*. Washington D.C.: Winston.
8. Anderson, J.R. and R. Thompson (1989): *Use of analogy is a production system architecture*, in *Similarity and Analogical Reasoning*, S. Vosniadou and A. Ortony, Editors. Cambridge University Press: Cambridge. p. 267-297.
9. Anick, P.G. (1993): *Integrating natural language processing and information retrieval in a troubleshooting help desk.*IEEE Expert. **8**(6): p. 9-17.
10. Arnold, S.P. and S.L. Stepoway (1988): *The reuse system: Cataloging and retrieval of reusable software*, in *Software Reuse: Emerging Technology*, W. Tracz, Editor. Computer Society Press: Washington, D.C. p. 138-141.
11. Balzer, R. (1985): *15 year perspective on automatic programming.* IEEE Trans. on Soft. Eng. **SE**(11): p. 1257-1268.
12. Balzer, R.M., N. Goldman, and D. Wile (1978): *Informality in program specifications.* IEEE Trans. on Software Eng. **SE**(4): p. 94-103.
13. Bellinzona, R., M.G. Fugini, and B. Pernici (1995): *Reusing specifications in OO applications.* IEEE Software. **12**(2): p. 65-75.
14. Berry, D.M., N.M. Yavne, and M. Yavne (1987): *Application of program design language tools to Abbott's method of program design by informal natural language descriptions.* Journal of Systems and Software. **7**: p. 221-247.
15. Biebow, B. and S. Szulman (1993): *Acquisition and validation: from text to semantic network.* in *Knowledge Acquisition for Knowledge-Based Systems*. Toulouse and Caylus, France: Springer-Verlag, p. 427-446.
16. Bigelow, J. (1988): *Hypertext and CASE.* IEEE Software: p. 23-27.
17. Bobrow, D.G. and T. Winograd (1977): *An overview of KRL, a knowledge representation language.* Cognitive Science. **1**: p. 3-46.
18. Brachman, R.J., D.L. McGuinness, P.F. Patel-Shneider, L.A. Resnick, and A. Borgida (1991): *Living with Classic: When and how to use a KL-ONE like language*, in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, J.F. Sowa, Editor. Morgan Kaufmann Pub., Inc.: San Mateo, California. p. 401-456.
19. Brachman, R.J. and J.G. Schmolze (1985): *An overview of KL-ONE knowledge representation system.* Cognitive Science. **9**(2): p. 171-216.
20. Burton, B.A., R.W. Aragon, S.A. Bailey, K.D. Koehler, and L.A. Mayes (1987): *The reusable software library.* IEEE Software. **4**(4): p. 25-33.
21. Buschmann, F. (1994): *Software architecture and reuse - An inherent conflict?* in *Third International Conference on Software Reuse: Advances in Software Reusability*. Rio de Janeiro, Brazil: IEEE Computer Society Press, p. 218-219.
22. Buschmann, F., R. Meunier, P. Sommerlad, and M. Stal (1996): *Pattern Oriented Software Architecture: A System of Patterns*. Chichester: JohnWiley and Sons.
23. Bush, V. (1945): *As we may think.* Atlantic Monthly. **176**: p. 101-108.

24. Carando, P. (1989): *Shadow: Fusing hypertext with AI.* IEEE Expert. **4**(4): p. 65-78.

25. Carlson, D.A. and S. Ram (1990): *HyperIntelligence: the next frontier.* Communications of the ACM. **33**(3): p. 311-321.

26. Castano, S. and V. De Antonellis (1994): *The F3 Reuse Environment for Requirements Engineering.* ACM SIGSOFT Software Engineering Notes. **19**(3): p. 62 -65.

27. Cima, A.M.d., C.M.L. Werner, and A.A.C. Cerqueira (1994): *The design of object-oriented software with domain architecture reuse*. in *Third International Conference on Software Reuse: Advances in Software Reusability*. Rio de Janeiro, Brazil: IEEE Computer Society Press, p. 178-187.

28. Cohen, S. (1997): *Object technology, architectures and domain analysis: What's the connection? Is there a connection?* ACM Software Engineering Notes(September/October): p. contribution to WISR-8: Summary and Working Groups Report by Stephen H. Edwards and Bruce W. Weide.

29. Conklin, J. (1987): *Hypertext: An Introduction and Survey.* IEEE Computer: p. 17-40.

30. Conklin, J. and M.L. Begeman (1987): *gIBIS: A hypertext tool for team design deliberation*. in *Hypertext'87*. Chapel Hill, NC: ACM, p. 247-251.

31. Creech, M.L., D.F. Freeze, and M.L. Griss (1991): *Using hypertext in selecting reusable software components*. in *Hypertext'91*. San Antonio, Texas: Acm, p. 25-38.

32. Croft, W.B. (1993): *Knowledge-based and statistical approaches to text retrieval.* IEEE Expert. **8**(2): p. 8-12.

33. Croft, W.B. (1995): *Effective text retrieval based on combining evidence from the corpus and users.* IEEE Expert. **10**(6): p. 59-63.

34. Cybulski, J.L. and K. Reed (1992): *A hypertext-based software engineering environment.* IEEE Software. **9**(2): p. 62-68.

35. D'Alessandro, M., P.L. Iachini, and A. Martelli (1993): *The generic reusable component: an approach to reuse hierarchical OO designs*, in *Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability*, P.-D. Ruben and B.F. William, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 39-46.

36. Dankel, D.D., M.S. Schmalz, and K.S. Nielsen (1994): *Understanding natural language software specifications*. in *Fourteen International Avignon Conference, AI'94*. Paris, France: Ec-2, p. .

37. DDC-Summaries (1989): *Dewpoint: The fast, well-organised Internet catalog*, Web Site http://ivory.lm.com/~mundie/DDHC/DDH.html, OCLC Online Library Center, Inc.

38. Deering, M., J. Faletti, and R. Wilensky (1982): *Using the PEARL AI Package*, , Uni of California, Comp. Sci. Division.

39. Deutsch, L.P. (1989): *Design reuse and frameworks in the Smalltalk-80 system*, in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. ACM Addison Wesley Publishing Company: New York, New York. p. 57-71.

40. Devanbu, P., R.J. Brachman, P.G. Selfridge, and B.W. Ballard (1991): *LaSSIE: A knowledge-based software information System.*Communications of ACM. **34**(5): p. 34-49.

41. Dillon, T.S. and P.L. Tan (1993): *Object-Oriented Conceptual Modeling*. Sydney: Prentice-Hall.

42. DoD (1995): *Software Reuse Initiative: Technology Roadmap, V2.2*, Report http://sw-eng.falls-church.va.us/reuseic/policy/Roadmap/Cover.html, Department of Defense.

43. Ellis, G.P., J.E. Finlay, and A.S. Pollit (1994): *HIBROWSE for hotels: bridging the gap between user and system views of a database*. in *2nd Int. Workshop on Interfaces to Database Systems*. Lancaster University, UK: Springer-Verlag, p. 49-62.

44. Embley, D.W. and S.N. Woodfield (1987): *A knowledge structure for reusing abstract data types*. in *Ninth International Conference on Software Engineering*. Monterey, CA: IEEE Computer Society Press, p. 360-368.

45. Engelien, B. and M. Ronnie (1991): *Natural Language Markets: Commercial Strategies*. London, England: Ovum Ltd.

46. Fikes, R. and T. Kehler (1985): *The role of frame-based representation in reasoning.* Communications of the ACM. **28**(9): p. 904-920.

47. Fillmore, C.J. (1968): *The case for case*, in *Universals in Linguistic Theory*, E. Bach and R.T. Harms, Editors. Holt, Rinehart and Winston: New York. p. 1-88.

48. Fowler, M. (1997): *Analysis Patterns: Reusable Object Models*. Menlo Park, California: Addison-Wesley.

49. Frakes, W.B. and B.A. Nejmeh (1988): *An information system for software reuse*, in *Tutorial on Software Reuse: Emerging Technology*, W. Tracz, Editor. IEEE Computer Society Press: Washington, D.C. p. 142-151.

50. Frakes, W.B. and T.P. Pole (1994): *An empirical study of representation methods for reusable software components.* IEEE Transactions on Software Engineering. **20**(8): p. 617-630.

51. Fuchs, N.E., H.F. Hofmann, and R. Schwitter (1994): *Specifying Logic Programs in Controlled Natural Language*, 94.17, Department of Computer Science, University of Zurich.

52. Fugini, M.G. and S. Faustle (1993): *Retrieval of reusable components in a development information system*, in *Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability*, P.-D. Ruben and B.F. William, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 89-98.

53. Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*. Readings, Massachusetts: Addison-Wesley.

54. Garg, P.K. and W. Scacchi (1989): *ISHYS: Designing an intelligent software hypertext system.* IEEE Expert. **4**(3): p. 52-63.

55. Garg, P.K. and W. Scacchi (1990): *Hypertext system to manage software life-cycle documents.* IEEE Software. **7**(3): p. 90-98.

56. Girardi, M.R. and B. Ibrahim (1994): *Automatic indexing of software artefacts*. in *3rd Int. Conf. on Software Reuse*. Rio de Janeiro, Brasil, p. 24-32.

57. Girardi, M.R. and B. Ibrahim (1994): *A similarity measure for retrieving software artefacts*. in *6th Int. Conf. on Software Engineering and Knowledge Engineering*. Jurmala Latvia, p. 478-485.

58. Goldberg, A. (1984): *The influence of an object-oriented language on the programming environment*, in *Interactive Programming Environments*, D. Barstow, H. Shrobe, and E. Sandewall, Editors. McGraw-Hill. p. 141-174.

59. Goldin, L. and D.M. Berry (1994): *AbstFinder, a prototype abstraction finder for natural language text for use in requirement elicitation: design, methodologies, and evaluation*. in *The First International Conference on Requirements Engineering*. Colorado Springs, Colorado: IEEE Computer Society Press, p. 84-93.

60. Goldstein, I.P. and B. Roberts (1979): *Using Frames in Scheduling*. in *Winston and Brown 1979*, p. 256-284.

61. Graham, I. (1994): *Object Oriented Methods*. Wokingham, England: Addison-Wesley Pub. Co.

62. Griss, M.L., C. Jette, W.V. Kozaczynski, R. Troy, and A.I. Wasserman (1994): *Object-Oriented Reuse*. in *Third International Conference on Software Reuse: Advances in Software Reusability*. Rio de Janeiro, Brazil: IEEE Computer Society Press, p. 209-213.

63. Griss, M.L. and M. Wosser (1995): *Making reuse work at Hewlett-Packard.* IEEE Software(January): p. 105-107.

64. Hadjami, H., B. Ghezala, and F. Kamoun (1995): *A reuse approach based on object-orientation. Its contribution in the development of CASE tools*. in *Symposium on Software Reusability*. Seattle, Washington: ACM Press, Software Engineering Notes, p. 53-62.

65. Hahn, U. (1989): *Making understanders out of parsers: Semantically driven parsing as a key concept for realistic text understanding applications.* International Journal of Intelligent Systems. **4**(3): p. 345-393.

66. Heidorn, G.E. (1976): *Automatic programming through natural language dialogue: A survey.* IBM J. Res. Develop. **20**(4): p. 302-313.

67. Henninger, S. (1994): *Using iterative refinement to find reusable software.* IEEE Software. **11**(5): p. 48-59.

68. Holyoak, K.J. (1990): *Problem solving*, in *An Invitation to Cognitive Science: Thinking*, D.N. Osherson and E.E. Smith, Editors. The MIT Press: Cambridge, Massachusetts. p. 117-146.

69. IMSL (1995): *The IMSL Product Family*, WWW Document http://www.vni.com/index.html, Visual Numerics, Inc: Houston, Texas.

70. Jacobs, P.S. (1993): *Using statistical methods to improve knowledge-based news categorisation.* IEEE Expert. **8**(2): p. 13-23.

71. Jacobson, I., M. Griss, and P. Jonsson (1997): *Software Reuse: Architecture, Process and Organization for Business Success*. New York, NY: Addison-Wesley.

72. Jarke, M., J.A. Turner, E.A. Stohr, Y. Vassiliou, N.H. White, and K. Michielsen (1985): *A field evaluation of natural language for data retrieval.* IEEE Transactions on Software Engineering. **SE**(11): p. 97-114.

73. Kaindl, H. (1993): *The missing link in requirements engineering.* ACM SIGSOFT Software Engineering Notes. **18**(2): p. 30-39.

74. Kaiya, H., M. Saeki, and K. Ochimizu (1995): *Design of a hyper media tool to support requirements elicitation meetings*. in *Seventh International Workshop on Computer-Aided Software Engineering*. Toronto, Ontario, Canada: IEEE Computer Society Press, Los Alamitos, California, p. 250-259.

75. Kuipers, B.J. (1975): *A frame for frames*, in *Representation and Understanding*, D.G. Bobrow and A. Collins, Editors. Academic Press, Inc.: New York, NY. p. 151-184.

76. Lee, H.-Y. and M.T. Harandi (1993): *An analogy-based retrieval mechanism for software design reuse*. in *KBSE'93, The Eighth Knowledge-Based Software Engineering Conference*. Chicago, Illinois: IEEE Computer Society Press, p. 152-159.

77. Lehnert, W. and B. Sundheim (1991): *A performance evaluation of text-analysis technologies.* AI Magazine. **12**(3): p. 81-94.

78. Lewis, D.D., W.B. Croft, and N. Bhandaru (1989): *Language oriented information retrieval.* International Journal of Intelligent Systems. **4**(3): p. 285-318.

79. Lewis, D.D. and K. Spark-Jones (1996): *Natural language processing for information retrieval.* Communications of the ACM. **39**(1): p. 92-101.

80. Liao, H.-C. and F.-J. Wang (1993): *Software reuse based on a large object-oriented library.* ACM SIGSOFT, Software Engineering Notes. **18**(1): p. 74-80.

81. Lung, C.-H. and J.E. Urban (1995): *An approach to the classification of domain models in support of analogical reuse.* ACM Software Engineering Notes. **Proc. Symposium of Software Reusability, SSR'95**: p. 169-178.

82. Maarek, Y.S., D.M. Berry, and G.E. Kaiser (1991): *An information retrieval approach for automatically constructing software libraries.* IEEE Transactions on Software Engineering. **17**(8): p. 800-813.

83. MacGregor, R. (1991): *The evolving technology of classification-based knowledge representation systems*, in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, J.F. Sowa, Editor. Morgan Kaufmann Pub., Inc.: San Mateo, California. p. 385-400.

84. MacKellar, B.K. and F. Maryanski (1989): *Knowledge base for code reuse by similarity*. in *Proc. COMPSAC 89*. Orlando, FL, USA: IEEE, IEEE Service Center, Piscataway, NJ, USA, p. 634-641.

85. Maida, A.A. and S.C. Shapiro (1982): *Intensional concepts in propositional semantic networks.* Cognitive Science. **6**(4): p. 291-330.

86. Maiden, N. and A. Sutcliffe (1991): *Analogical matching for specification reuse*. in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 108-116.

87. Meyer, B. (1987): *Reusability: the case for object-oriented design.* IEEE Software: p. 50-64.

88. Microsoft (1992): *Microsoft C/C++, Version 7.0*, , Microsoft.

89. Mili, H., E. Ah-Ki, R. Godin, and H. Mcheick (1997): *Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval.* Software Engineering Notes. **22**(3): p. 89-98.

90. Mili, H., H. Sahraoui, and I. Benyahia (1997): *Representing and querying reusable object frameworks.* Software Engineering Notes. **22**(3): p. 110-120.

91. Minsky, M. (1975): *A framework for representing knowledge*, in *The Psychology of Computer Vision*, P. Winston, Editor. McGraw-Hill: New York. p. 211-280.

92. Miriyala, K. and M.T. Harandi (1991): *The role of analogy in specification derivation*. in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 117-126.

93. Moineau, T., J. Abadir, and E. Rames (1990): *Towards generic and extensible reuse environment*. in *Software Engineering 90*. Brighton, U.K.: Cambridge University Press, p. 543-573.

94. Naka, T. (1987): *Pseudo Japanese specification tool.* Faset. **1**: p. 29-32.

95. Nanduri, S. and S. Rugaber (1995): *Requirements validation via automatic natural language parsing.* Journal of Management Information Systems. **12**(2): p. 9-19.

96. Nielsen, J. (1990): *The art of navigating through hypertext.* Communications of the ACM. **33**(3): p. 296 -310.

97. Ostertag, E., J. Hendler, R. Prieto-Diaz, and C. Braun (1992): *Computing similarity in a reuse library system: An AI-based approach.* ACM Transactions on Software Engineering and Methodology. **1**(3): p. 205-228.

98. Pollit, A.S., G.P. Ellis, and M.P. Smith (1994): *HIBROWSE for bibliographic databases.* Journal of Information Science. **20**(6): p. 413-426.

99. Pressman, R.S. (1992): *Software Engineering: A Practitioner's Approach*. 3 ed. New York, N.Y.: McGraw-Hill, Inc.

100. Prieto-Diaz, R. (1989): *Classification of reusable modules*, in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. Addison-Wesley Pub. Co.: New York, NY. p. 99-123.

101. Prieto-Diaz, R. (1991): *Implementing faceted classification for software reuse.* Communications of ACM. **34**(5): p. 88-97.

102. Prieto-Diaz, R. and P. Freeman (1987): *Classifying software for reusability.*IEEE Software. **4**(1): p. 6-16.

103. Prieto-Diaz, R. and G.A. Jones (1987): *Breathing new life into old software.* GTE Journal of Science and Technology. **Spring**: p. 23-31.

104. Quillian, M.R. (1968): *Semantic memory*, in *Semantic Information Processing*, M. Minsky, Editor. MIT Press: Cambridge, MA. p. 227-270.

105. Ranghanathan, S.R. (1957): *Prolegomena to Library Classification*. Letchworth, Hertfordshire, UK: The Garden City Press Ltd.

106. Reed, K. (1992): *Successful large-scale reuse in Cobol, Fortran, C and Ada*, , La Trobe University.

107. Roberts, R.B. and I.P. Goldstein (1977): *The FRL Primer*, , MIT, AI Lab.

108. Rolland, C. and C. Proix (1992): *A Natural Language Approach For Requirements Engineering*. in *Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*. Lecture Notes in Computer Science, Vol. 593: Springer Verlag, p. 257-277.

109. Saeki, M., H. Horai, and H. Enomoto (1989): *Software development process from natural language specifications*. in *11th International Conference on Software Egineering*. Pittsburgh, Pennsylvania: IEEE Computer Press, p. 64-73.

110. Salek, A., P.G. Sorenson, J.P. Tremblay, and J.M. Punshon (1994): *The REVIEW system: From formal specifications to natural language*. in *The First International Conference on Requirements Engineering*. Colorado Springs, Colorado: IEEE Computer Society Press, p. 220-229.

111. Salton, G. (1989): *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Readings, Massachusetts: Addison-Wesley Pub. Co.

112. Salton, G., C. Buckley, and M. Smith (1990): *On the application of syntactic methodologies in automatic text analysis.* Information Processing & Management. **26**(1): p. 73-92.

113. Schank, R.C. and K.M. Colby (1973): *Computer Models of Thought and Language*: Freeman & Co.

114. Schwitter, R. and N.E. Fuchs (1996): *Attempto - from specifications in controlled natural language towards executable specifications*. in *GI EMISA Workshop, Natürlichsprachlicher Entwurf von Informationssystemen*. Tutzing, Germany, p. 163-177.

115. Sembok, T.M.T. and C.J. Van Rijsbergen (1990): *SILOL: a simple logical-linguistic document retrieval system.* Information Processing & Management. **26**(1): p. 111-134.

116. Shapiro, S.C., ed. (1987): *Encyclopaedia of Artificial Intelligence*. . John Wiley & Sons: New York, N.Y.

117. Smeaton, A.F. (1989): *Information Retrieval and Natural Language Processing*. in *Informatics 10: Prospects for Intelligent Retrieval*. Cambridge, England: Aslib, The Association of Information Management, p. 1-14.

118. Sorumgard, L.S., G. Sindre, and F. Stokke (1993): *Experiences from application of a faceted classification scheme*, in *Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability*, P.-D. Ruben and B.F. William, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 116-124.

119. Sowa, J.F. (1984): *Conceptual Structures: Information Processing in Mind and Machine*. Readings, Massachusetts: Addison-Wesley Pub. Co.

120. Vadera, S. and F. Meziane (1994): *From English to formal specifications.* The Computer Journal. **37**(9): p. 753-763.

121. Vosniadou, S. and A. Ortony (1989): *Similarity and analogical reasoning: a synthesis*, in *Similarity and Analogical Reasoning*, S. Vosniadu and A. Ortony, Editors. Cambridge University Press: Cambridge. p. 1-17.

122. Wilks, Y. (1975): *An inteligent analyzer and understander of English.* Communications of the ACM. **18**(5): p. 264-274.

123. Winston, P.H. (1975): *Learning structural descriptions from examples*, in *The Psychology of Computer Vision*, P.H. Winston, Editor. McGraw-Hill Book Co.: New York, NY. p. 157-209.

124. Winston, P.H. (1980): *Learning and reasoning by analogy.* Communications of the ACM. **23**(12): p. 689-703.

125. Wirfs-Brock, R.J. and R.E. Johnson (1990): *Surveying current researc h in object-oriented design.* Communications of the ACM. **33**(9): p. 104-124.

126. Woods, W.A. (1991): *Understanding subsumption and taxonomy: A framework for progress*, in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, J.F. Sowa, Editor. Morgan Kaufmann Pub., Inc.: San Mateo, California. p. 45-94.

127. Young, M. and K. Reed (1992): *Identifying reusable components in software requirements specifications to develop a natural language-like SRS language with a CRNLP*, Technical Report TR005, Amdahl Australian Intelligent Tools Programme: Bundoora, Vic, Australia.