# Cataloging Design Abstractions

Spencer Rugaber
College of Computing
Georgia Institute of Technology
Atlanta, Georgia USA
+1 404.894.8450
spencer@cc.gatech.edu

*"He, whose design includes whatever language can express, must often speak of what he does not understand." — Samuel Johnson*

## ABSTRACT

Abstractions are the essence of software design, and various enterprises, such as design patterns, architectural styles, programming clichés and idioms, attempt to capture, organize and present them to software engineers. This position paper explores the possibility of mounting a more comprehensive effort to catalog abstractions. Related efforts such as the design of textual and electronic dictionaries, markup languages for software artifacts and ontologies of computer science topics are surveyed to inform the effort. A set of derivative questions is presented to explore the problem space.

## Categories and Subject Descriptors

D.2.7 [**Software Engineering**] Distribution, Maintenance and Enhancement – *Documentation, Restructuring, reverse engineering, and reengineering*.

D.3.3 [**Programming Languages**] Language Constructs and Features – *Patterns*.

K.3.2 [**Computers and Education**] Computer and Information Science Education – *Computer Science Education*.

## General Terms

Documentation, Design.

## Keywords

Abstraction, architectural styles, computer science education, design patterns, programming idioms, program understanding, reverse engineering

## 1. INTRODUCTION

Mary Shaw, in describing the maturity of software engineering as a discipline, has pointed out [24] that "software lacks the institutionalized mechanisms of a mature engineering discipline for recording and disseminating demonstrably good designs and ways to choose among design alternatives". Since the time of Shaw's article, software researchers have responded in a variety of ways. Gamma *et al.* and their successors have produced a wide collection of resources exploring the concept of *patterns* (design patterns [9], analysis patterns [7], reengineering patterns [5] and even anti-patterns [3]). The software architecture research community has explored the concept of *styles* of software architectures [25]. Program understanding researchers have collected and cataloged programming *clichés* [29] and *idioms* [17]. Each of these approaches provides a subset of the vocabulary of abstractions that a designers use in building programs. Shaw, referring to the work of Herbert Simon, defines the target: "An expert in a field must know about 50,000 chunks of information, where a chunk is any cluster of knowledge sufficiently familiar that it can be remembered rather than derived. Furthermore, in domains where there are full-time professionals, it takes no less than 10 years for a world-class expert to achieve that level of proficiency."

We are concerned with supporting designers and programmers, which we call "designers" in the remainder of the paper, as they develop software. By *abstraction*, we mean a domain-independent unit of a design vocabulary that subsumes more detailed information. This paper addresses the question of whether is it is possible to construct a catalog (a *plexicon* or programmer's lexicon) of such abstractions, and, if so, what form should such a catalog take?

A catalog of abstractions would have a variety of benefits.

- Developers trying to solve design problems would have a resource similar to the dictionaries and thesauri that writers use;
- Software maintainers and reverse engineers would know what they were looking for. Reverse engineering tools would have explicit targets;
- Students trying to learn design would have a reference source;
- The process of building and populating such a repository would itself uncover knowledge of the nature of abstractions.

This position paper first examines a wide variety of related efforts. These include general mechanisms, such as dictionaries, both printed and electronic, and support technology, such as markup languages, databases and ontologies, that can be applied to cataloging design abstractions. The paper then discusses the implications of this work on the construction of a plexicon.

## 2. RELATED WORK

Shaw points out that mature engineering disciplines often have handbooks that collect much of the knowledge of that discipline in one place. Handbooks can provide an organizational structure for the discipline, short articles describing individual units of knowledge, mathematical and scientific foundations such as formulae and tables, process descriptions, evaluation techniques, glossaries and references to other resources.

Currently, the primary reference source for software designers is the traditional written text. Notable are Knuth's seminal works on algorithms [11] and the *Numerical Recipes* books (`www.nr.com`). But we can learn also from other efforts. Abstractions form a vocabulary, and Sections 2.1, 2.2 and 2.3 describe the content and organization of natural language dictionaries and thesauri and their electronic successors. Existing efforts to provide on-line access to design information are presented in Section 2.4, and Table 1 summarizes the variations among them. Section 2.5 summarizes two efforts that use markup languages to describe software artifacts for electronic distribution. Finally, Section 2.6 presents other technologies that might be applied to support the automated access and management of design knowledge.

## 2.1 Natural Language Dictionaries

The word *dictionarius* was first used in the 13th century, and English language dictionaries have existed for more than four hundred years. Landau [13] gives a fascinating presentation not only of the history of dictionaries, but also describes the many design issues involved in constructing them. Examples of such issues include whether a dictionary is mono- or multi-lingual, the age of the intended user, the size (number of entries) and extent of coverage, the subject area covered, the period of history being covered (*e.g.* Middle English), the purpose of the dictionary (increasing vocabulary, presenting knowledge, providing etymologies), the lexical unit covered (words, terms, phrases), the attitude toward standardization (that is, whether the dictionary is drawn from actual usage or is intended normatively), the means of access (print or electronic), the dictionary's tone (detached, didactic, facetious), its organization (alphabetical, by sound, by concept), the documentation provided (historical notes and reference sources), the contents of an entry (orthography, pronunciation, senses, definitions, inflected forms, synonyms, usage examples, differentiation from related concepts, usage guidance), and any special features provided (proper names, abbreviations, etc.).

The essence of a dictionary is, of course, its entries. Each entry can include a wide variety of information including spelling; hyphenation; variants; homographs; pronunciation; syllable divisions; stress; part of speech; prefixes; suffixes; combining forms; plurals; tenses; participles; comparatives; superlatives; capitalization; etymology; status labels: temporal (*obsolete*, *archaic*), stylistic (*slang*, *substandard*, *non-standard*) and regional (*dialect* or specific region or country); definition; senses; illustrations; examples; usage notes; cross references; and inflected forms.

And there is more. The Explanatory Notes section of a traditional dictionary, such as [28], also includes the set of principles (process rules) used to actually construct an entry. In [28], the rules specify that the dictionary be self contained (all words in definitions should have their own entries), that the words used in defining a concept be simpler than the concept being defined, that the definitions be non-circular and definitive, and that the phrasing of the definition correspond to the part of speech of the word being defined.

The above lists provide the functional requirements for a dictionary. The most significant issue is, however, non-functional. Dictionaries are compared by the number of entries they contain, but their costs are proportional to their size in pages. Therefore, the editor's key job is to pack as many entries in as few pages as possible while including the details mentioned above. Thought of in this way, it can be seen that dictionary construction is itself a design problem.

## 2.2 Thesauri

Related to dictionaries, but serving a somewhat different role, are thesauri. Although general thesauri, such as Roget's [22], exist, most often they cover only a specific topic area, and, more importantly, they are intended normatively; that is, they try to act as a standard vocabulary for that area. A thesaurus can include information on preferences between synonymous terms, related terms and usage guidance. The key difference from dictionaries is that the entries in a thesaurus are organized conceptually rather than alphabetically. Hence, the conceptual organization is itself a design activity.

A thesaurus is a "vocabulary of controlled indexing language formally organized so that a priori relationships between concepts are made explicit" [1]. Like dictionaries, there is a long history demonstrating much variation. Consequently, the design space for thesauri is quite rich. Ultimately, a thesaurus, like a dictionary is an index into a set of knowledge entries. The chief choice in thesaurus design is whether the indexing vocabulary is controlled or natural. With a controlled vocabulary, search precision is improved at the cost of the effort required to select and organize the vocabulary. With automated analysis of free text, statistical techniques can be used to make natural vocabularies competitive with controlled ones. The most fundamental relationship that a thesaurus supports is the mapping between the index terms and the underlying material being indexed. Other relationships supported by thesauri include synonyms, broader and narrower terms, associated terms, whole–parts, and instances.

There are a large collection of design choices that a thesaurus editor can select from. Among these are of course the corpus being indexed, the indexing language (mono or multi-lingual, natural or controlled), search key format (stemmed, part of speech, Boolean connectives), the size relationship between the number of items being indexed and the number of indexing terms, pre and post processing on the search request, textual and graphical presentation of the corpus, and presentation options for search results.

## 2.3 Electronic Dictionaries

In addition to traditional printed media, dictionary purveyors and researchers have explored the use of electronic presentation. Notable efforts include the following.

- Providing a hypertext version of the *Oxford English Dictionary (OED)* [19]. "The Oxford English Dictionary is the largest and most scholarly dictionary of written English." The cited paper examines the question of how the OED might be converted to hypertext form. Motivation for hypertext includes support for browsing, providing alternative forms for displaying entries and making the dictionary a better match for its users' tasks. The effort also involves interfacing the resulting hypertext to other automated tools. The key questions that arise are what exactly are the nodes (that is, what are the targets of links), what is the nature of the tags used and what kinds of links should be employed. Examples of the latter might include links between the words used in a definition and their entries, links to synonyms, explicit cross references, links to variants and even, it might be imagined, links from example uses to an electronic version of the material from which they were taken. Other issues that arise relate to use, such as the ability to save results, to store and reuse queries, to add annotations, and to provide additional material, generate reports, and filter retrievals. The current on-line version of the OED (`www.oed.com/about/oed-online`) includes control over display of entries; Boolean, wild-card queries; lookup by meaning, language of origin, source or year of entry; and limited hyperlinked cross references.

- WordNet (`wordnet.princeton.edu`) [15] provides an interesting contrast with the OED. "WordNet is an online lexical database designed for use under program control. English nouns, verbs, adjectives and adverbs are organized into sets of synonyms, each representing a lexicalized concept. Semantic relations link the synonym sets." In addition to hypertext links for traditional synonyms, WorldNet supports a variety of other relationships including antonyms; hyponyms (subordinates) and hypernyms (superordinates); meronyms (parts) and holonyms (wholes); troponomy (manner); and entailment. WordNet also provides an application program interface (API) so that other tools can readily access its content.
- EDR (`www.iijnet.or.jp/edr`) [31] is another electronic dictionary effort, this one originating in Japan. "The EDR Electronic Dictionary is a machine-tractable dictionary that catalogues the lexical knowledge of Japanese and English." Besides a dictionary of words, EDR includes a bilingual dictionary (Japanese-English), a co-occurrence dictionary to better understand phrasing, a concept dictionary (thesaurus), and a corpus database taken from published documents such as newspapers, to which the other dictionaries refer for usage information.
- Cyc (`www.cyc.com`) [14] is an even more ambitious project, attempting to electronically encode the knowledge needed to perform everyday tasks such as understanding newspaper articles. Its encoding is more formal than those systems described above enabling inferencing to be performed. The formal mechanism is intended to facilitate Cyc's use by other programs. Cyc comprises an extensive knowledge base, the inference engine, an underlying formal representation language, natural language processing technology, and API tools for third-party developers.

## 2.4 On-Line Dictionaries of Programming Concepts

Of course, many web-based repositories of design information already exist. Among the most interesting are the following.

- "Free Online Dictionary of Computing" (`www.foldoc.org`) from Imperial College's Department of Computing "is a searchable dictionary of acronyms, jargon, programming languages, tools, architecture, operating systems, networking, theory, conventions, standards, mathematics, telecoms, electronics, institutions, companies, projects, products, history, in fact anything to do with computing". Access is via keyword search with some cross-reference links. An index via first letter is provided into a page containing all terms beginning with that letter.
- Webopedia (`www.webopedia.com`), maintained by internet.com, provides keyword, category, and cross-reference access to computer-related terms and their definitions.
- The "Dictionary of Algorithms and Data Structures" (`www.nist.gov/dads`) from the National Institute of Standards and Technology "is a dictionary of algorithms, algorithmic techniques, data structures, archetypical problems and related definitions." Access methods include keyword search, alphabetical index, area index and category index. Some entries link to implementations, and there is a separate index of these implementations.
- "Algorithms and Data Structures Research & Reference Material" (`www.csse.monash.edu.au/~lloyd/tildeAlgDS`) from Monash University provides information about basic algorithms and data structures. Access is topical, with a separate presentation of implementations, and is organized by programming language. Cross references link entries to each other and to implementations.
- "Web Dictionary of Cybernetics and Systems" (`pespmc1.vub.ac.be/ASC/INDEXASC.html`) is hosted by the Principia Cybernetica project and includes alphabetically arranged concepts related to cybernetics. There is also a keyword search mechanism, cross-references and sequential links between alphabetically adjacent items.
- "The Stony Brook Algorithm Repository" (`www.cs.sunysb.edu/~algorith`) hosted by the State University of New York at Stony Brook is "a comprehensive collection of algorithm implementations for over seventy of the most fundamental problems in combinatorial algorithms." Access methods include keyword search, site outline, cross referencing, up/down/next/previous links and an image map. The site also supports user-contributed annotations.
- The University of Aukland, Computer Science Department, "Data Structures and Algorithms" (`www.cs.auckland.ac.nz/software/AlgAnim/ds_ToC.html`) contains introductory computer science material organized via a topical outline. Cross-references are included as well as implementations and animations. Forward-only links to the next topical entry are also provided.
- `hillside.net/pattern` is a website for pattern resources. It is informally organized as a set of links, both in-site and out-of-site. Some of the links are to diagrams for patterns from the Gamma *et al*. book [9].

Table 1 summarizes information about the sites in the electronic repositories.

**Table 1. On-Line Computer Science Dictionaries**

| | |
|---|---|
| **Contents** | Terminology, acronyms, concepts, implementations, animations, diagrams |
| **Organizational Mechanisms** | Topical outlines, alphabetical indexes, next/previous links, image map, cross-reference links, categories |
| **Target Audiences** | Beginning students, advanced students, practitioners |
| **Topic Areas** | Algorithms, data structures, cybernetics, idioms, patterns |
| **Sources** | User contributions, course materials, other dictionaries |
| **Other Features** | Programming language specificity, visitor annotation |

Note that the list of web-based dictionaries in this subsection does not include other organized collections of programming resources such as subprogram libraries, courseware and program analysis tools.

## 2.5 Software Resource Markup Languages

The previous subsections describe efforts to provide relatively informal organization and access to collected material. This section, in contrast, discusses two efforts that make use of markup languages to do the organizing.

- The Open Software Description Format (`www.w3.org/TR/NOTE-OSD.html`): "The goal of the OSD format is to provide an XML-based vocabulary for describing software packages and their inter-dependencies, whether it is user initiated ('pulled'), or automatic ('pushed')." The OSD vocabulary can be used in a stand-alone XML document to declare dependencies between different software components for different operating systems and languages, can accompany archive files, can convey the interdependency graph for the different software modules and can support automated distribution of components. Its basic organizational principle is a tree of component dependencies.
- Architectural Description Markup Language (`www.opengroup.org/architecture/adml/adml_home.htm`): This markup language is intended to describe architectural components for retrieval and testing. It is based on ACME, a notation for communicating between architecture tools. The web site above indicates that "ADML adds to ACME a standardization representation, the ability to define links to objects outside the architecture (such as rationale, designs, components, etc.), straightforward ability to interface with commercial repositories, and transparent extensibility."

## 2.6 Knowledge Organization

One of the key issues in a catalog of abstractions is how the contained knowledge will be organized. There are a variety of knowledge management mechanisms that can inform the design of a plexicon. In addition to the traditional dictionary and thesaurus, approaches include hierarchical (taxonomic) and faceted classifications, relational or object-oriented databases, full-blown knowledge bases (ontologies), and various mechanical approaches such as cluster analysis and concept hierarchies.

- **Taxonomies:** A *taxonomy* is a formal classification of a set of concepts. Normally, the classification is hierarchical and can either be tree-like (each entry has a single parent entry) or graph-like (where multiple parent entries are allowed). Both ACM (`www.acm.org/class/1998`) and IEEE (`www.computer.org/mc/keywords/software.htm`) have taxonomies of computer-related terminology for purposes of characterizing published articles.
- **Controlled vocabularies and faceted classification:** A *controlled vocabulary* is a set of terms used to index into a knowledge repository. The terms are carefully defined and static. If the terms are partitioned into orthogonal subsets (*facets*) that describe different aspects of the knowledge, then the knowledge repository is said to have a *faceted classification*. Faceted classifications, developed to support information retrieval, have been used for software reuse and application domain analysis [18]. One interesting controlled vocabulary providing access to software related assets is that provided by the U.S. Patent Office (`www.uspto.gov/patft/help/help.htm`). Patent records are described with about thirty-five primary fields that may be used in a search. Of primary interest is the Classification field, which, in turn, has about five hundred possibilities. Category 717 is the Software category that contains over one hundred keyword-based, hierarchically organized subdivisions. Recently, the open-source movement, in the form of various vendors and the Open Source Development Laboratory (`www.osdl.org`), has announced the creation of a related on-line repository of open source software assets, the Patent Commons Project (`www.patentcommons.org`).
- **Databases:** The results of analyzing programs have been stored in databases in a variety of ways. Perhaps the most frequent choice is to use a predefined schema to store the results in a relational database, as was done with the C Information Abstractor [4]. This approach can be contrasted with the use of an object-oriented database, such as the one provided by the Software Refinery [20] or any of a number of commercial systems. It should be noted of course, that UML [1] could be directly used as a representation for expressing design knowledge. Its textual extension, the Object Constraint Language [27] (OCL), can be used as a query language for accessing encoded design information. The tradeoff between the two database approaches seems to be the facile querying available relationally and the more flexible connectivity found with objects. One other approach should be noted; the Programmer's Apprentice Project [21] devised a means for representing programming knowledge in the form of *plans*. Their approach was called the Plan Calculus and featured a custom representation comprising a graph annotated by constraints.
- **Ontologies:** An *ontology* is a formal description of a vocabulary, typically including a set of concepts and the relationships among them. The formality enables machine representations and automated reasoning. Ontologies are a maturing technology that forms an essential part of the Semantic Web. Various representation languages have been developed and corresponding repositories populated.

Examples include KIF (www-ksl.stanford.edu/knowledge-sharing/kif), which includes ontologies for basic data structures such as lists and sets; Ontolingua (www.ksl.stanford.edu/software/ontolingua) and its follow-on KSL (www-ksl-svc.stanford.edu:5915), which provides an interactive ontology server; KQML (www.cs.umbc.edu/kqml) and DAML, together with its partner OIL (www.daml.org). Tool support for designing and maintaining ontologies includes knowledge representation languages, graphical editing environments and inference mechanisms. Examples include Protégé (protege.stanford.edu), PowerLOOM (www.isi.edu/isd/LOOM/PowerLoom), and Classic (www.research.att.com/sw/tools/classic), all of which have been used to represent software design information.

- **Mathematical approaches:** If a collection of elements being characterized is thought of as attribute–value pairs, then automated means can be used to characterize higher-order *concepts*. Concept analysis [26] is an approach in which the occurrence of common attributes is taken as evidence of a concept. Some concepts subsume others. Automated tools can construct elaborate lattices of shared attributes. Cluster analysis, in contrast, is statistical in nature [10]. Here each attribute serves as an axis in a multidimensional vector space. Each element then occupies a position in the space based on the values of its attributes. Cluster analysis attempts to group related elements into higher-order units based on their closeness (similarity) in this space. Both concept analysis and cluster analysis attempt to abstract higher-level understanding from constituent properties, and both have been used to subdivide complex software systems into their constituent artifacts.

## 3. ISSUES RAISED

The above survey raises many issues for consideration in designing a plexicon. Below we list several of these along with some of the possible answers.

- *What software engineering tasks should a plexicon support?* Manual entry and editing, annotation, automated capture (pattern detection, idiom extraction), keyword search, automated exploration via formal pattern matching and extensive cross-referenced exploration would all be of value.
- *What is it that is actually being cataloged?* The term *abstraction* is itself quite abstract. In addition to the definition given earlier in the paper, the Wikipedia (wikipedia.org) offers the following definition: "An abstraction is an idea, conceptualization, or word for the collection of qualities that identify the referent of a word used to describe concrete objects or phenomena." WordNet, (www.wordnet.org) in contrast, states only that *abstraction* is "a concept or idea not associated with any specific instance". Among possible constituents of a plexicon are terminology, in the sense of a traditional dictionary, algorithms and data structures, patterns, architectural styles, programming clichés and idioms, textbook examples, and programming language devices. Although the final arbiter will be usefulness, clearly an editorial inclusion policy is necessary.
- *What should be included in an entry?* The design patterns community has a standard format for the representation of patterns, including textual descriptions, UML diagrams and code samples. To this could be added indexing information, such as would be required for access via a controlled vocabulary. Likewise, provenance information should be included for benefit of the historical record and authenticity. A more formal specification, such as constitutes an ontology, could also be of value.
- *How should the data be organized? What underlying representation should be used?* All of the following have advantages. Relational databases support powerful tabular querying. An object-oriented representation would be compatible with UML/OCL-based tools. Marked-up (XML) text would enable participation in the Semantic Web. A more graph-based representation with formal annotations, such as the Programmer's Apprentice, would support pattern matching with existing code for plan recognition. Use of an ontology representation language would support formal reasoning.
- *What sorts of relationships should be supported?* A plexicon contains "chunks" of design knowledge that range in scope from architectural styles, through patterns to programming idioms. The basic organizational unit is the abstraction, and abstractions can be characterized in various ways [23]. The categories are derived from examining three areas of computer science: programming language design, data modeling and transformational programming. In all three areas, the following devices are identified, possibly using different names.
    - **Composition:** providing a single name that identifies a collection of subordinate elements. For example, in programming languages, a record structure abstracts a set of fields, and a subprogram aggregates a set of statements.
    - **Generalization:** characterizing one collection of instances as being a superset of another. In data modeling, this is sometimes called *superclassing*.
    - **Procedure/data:** alternatively considering a collection of data and the algorithm that produced it as equivalent. An example of this distinction is the classic time/space tradeoff. In transformational programming, memoization performs exactly the role of converting time-consuming computations into data accesses.
    - **Encapsulation/interleaving:** the systematic hiding of details beyond a formal barrier as contrasted with the intermixing of elements, usually to improve execution efficiency. This distinction is relevant to how an abstraction is expressed in an actual program.
    - **Representation:** the use of different constructs to express the same underlying concept. For example, a stack can be represented by a linked list or by an array plus an index. This can be thought of as a synonym.
    - **Non-determinism removal:** adding more constraints to bring a specification closer to an implementation. Implementing the stack from the previous item as a fixed-length array bounds the depth of the stack.

- *What is the role of formalism and inferencing in a plexicon?* Programs are ultimately formal objects, and abstractions over programs can benefit from precise formulation. Formality and reasoning can support detection, summarization, comparison and data mining.
- *What forms of presentation and data export should be used?* Possibilities include textual output with included graphics, code extraction with automated translation into various programming languages, XML, formal propositions suitable for input into an inference engine, overview indexes, summaries and graphical presentation of the entire space. In addition, a filtering mechanism and a report-writing capability would be of value.
- *What to do about programming language specificity?* Some abstractions are clearly more important to a particular programming language. Many uses for a plexicon will be in the context of a specific programming situation. It is therefore desirable to support direct expression of examples from particular languages. On the other hand, many abstractions derived from specific language context can be valuable when translated into other languages. It is interesting to contemplate the extent to which abstractions could be automatically translated among languages.
- *What editorial process is appropriate?* One model is an open forum, such as the Wikipedia (`wikipedia.org`), with anyone free to add material. Editorial policy includes an emphasis on consensus, including enforcement of agreed-upon principles such as conformance to its encyclopedic goal, avoidance of bias, adherence to copyright, and respect for other contributors. At the other end of the spectrum is construction by a single individual editor, perhaps with support of volunteers, such as was used for the *OED* [30]. The former can grow more quickly, but some of the advantages of consistency and uniformity may be lost.
- *How should the repository be populated?* The true benefit of a plexicon will arise only when it has obtained a critical mass of entries. It is therefore important to expedite its construction. Several sources of material come to mind. First, existing repositories can be harvested to the extent that legal access can be obtained. Second, if the goal is to fill a plexicon with the 50,000 chunks of knowledge required of an expert, then it would be of value to mimic the process by which the expert obtained the knowledge. One intriguing possibility is to follow the course of study of incoming Computer Science students, archiving the abstractions they obtain from textbooks, lectures, and exercises. The third possibility for population is via reverse engineering; that is, the systematic examination of existing programs for the purpose of cataloging the constituent abstractions. A fourth, speculative, possibility is to extract common abstractions statistically by examining frequently occurring mechanisms, appropriately abstracted from their program settings. A final possibility is manual construction via volunteers, where the community itself provides much of the editorial oversight.
- *How might a plexicon support Computer Science education?* Clearly, an appropriately organized catalog of abstractions would be of value to Computer Science education. We have experimented with the construction of a small plexicon (`www.cc.gatech.edu/projects/ plexicon`) in support of a single, sophomore-level course, CS2130, *Languages and Translation*. The plexicon was limited in scope, comprising about ninety entries, and power. Entries contained the following fields: Name, Keywords (uncontrolled), Category, See Also (cross-references), Aggregates (meronyms) and Aggregate Of (holonyms), Specializations (hyponym) and Generalizations (hypernyms), Explanation, When to Use (context), Examples, Contributor, Citation, and Last Modified. Despite the limitations, the students found it useful, particularly in preparation for their exams. It is also interesting to contemplate the feedback that might be obtained from viewing a history of student interactions. Particularly useful would be learning about the connections the students traversed between entries and the dwell time for particular entries.
- *What other interesting applications could benefit from a plexicon?* Intriguing ideas include intelligent patent search, retrieval from reuse libraries, use of the abstractions by a program generation tool and use in random program construction, *a la* genetic programming [12].
- *What research possibilities relate to the cataloging of abstractions?* The construction of a catalog of abstractions will itself lead to interesting research results, just as the work with design patterns and other targeted abstractions has. Foremost is obtaining an empirical understanding of what the space of abstractions is and how it is organized. Are some abstractions more error prone than others? What is the relationship between abstractions and programming languages? How, within specific programs, are abstractions composed? Within programs, what is the relationship between programming abstractions and non-abstraction code? How do abstractions relate to design refactorings [8] and other program transformations [16]?

## 4. CONCLUSIONS

As Shaw points out, a handbook of design knowledge will be a coming-of-age demonstration for software engineering. I see this handbook taking the form of a catalog of abstractions. This position paper explores relevant background to constructing such a catalog of design vocabulary and discusses issues in its construction.

The Danish design researcher, Pelle Ehn, likewise stresses the importance of vocabulary. He relates the work of the philosopher Ludwig Wittgenstein to the software design process [6]. Wittgenstein explored the specialized vocabulary used between crafts people in constructing artifacts, such as buildings. He called these specialized vocabularies *language games*. Ehn contrasts Wittgenstein's approach to design with that of Descartes, which is based on analysis and "rationalistic reasoning". In particular, Ehn stresses two aspects related to plexicons: the importance of specialized vocabulary and its foundation in actual use. Both of these aspects are, of course, historically central to the construction of dictionaries.

> *"By understanding design as a process of creating new language-games that have family resemblance with the language-games of both users and designers we have an orientation for really doing design as skill based participation, a way of doing design that may help us to*

*transcend some of the limits of formalization." — Pelle Ehn*

## 5. REFERENCES

[1] Aitchison, J. and Gilchrist, A. *Thesaurus Construction, Second Edition.* Aslib, The Association for Information Management, London, England, 1987.

[2] Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide.* Addison Wesley, 1999.

[3] Brown, W. J., Malveau, R. C., McCormick, H. W., and Mowbray, T. J. *AntiPatterns.* John Wiley, 1998.

[4] Yih-Farn Chen, Y-F., Nishimoto, M. Y., and Ramamoorthy, C. V. The C information abstraction system. *IEEE Transactions on Software Engineering, 16,* 3, (Mar. 1990), 325-334.

[5] Demeyer, S., Nierstrasz, O. M., Ducasse, S. *Object-Oriented Reengineering Patterns.* Morgan Kaufmann, 2002.

[6] Ehn, P. Playing the language-games of design and use-On skill and participation. In the *Conference on Supporting Group Work Archive,* ACM, (Palo Alto, California), 1988, 142-157.

[7] Fowler, M. *Analysis Patterns: Reusable Object Models.* Addison-Wesley, 1996.

[8] Fowler, M. *Refactoring*. Addison Wesley, 1999.

[9] Gamma E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Abstraction and Reuse of Object-Oriented Software.* Addison-Wesley, 1995.

[10] Jain, A. K., Murty, M. N. and Flynn, P. J. Data clustering: A review. *ACM Computing Surveys, 31,* 3, (Sep. 1999), 264-323.

[11] Knuth, D. E. *The Art of Computer Programming, Volumes 1-3.* Addison Wesley, 1998.

[12] Koza, J. R. *Genetic Programming / On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[13] Landau, S. I. *Dictionaries / The Art and Craft of Lexicography.* Charles Scribner, 1984.

[14] Lenat, D. B.. Cyc: A large scale investment in knowledge infrastructure. *Communications of the ACM, 38,* 11, (Nov. 1995), 33-38.

[15] Miller, G. A. WordNet: A lexical database for English. *Communications of the ACM, 38,* 11, (Nov. 1995), 39-41.

[16] Partsch, H. and Steinbruggen, R. Program transformation systems. *ACM Computing Surveys, 15,* 3, (Sep. 1983), 189-226.

[17] Perlis, A. and Rugaber, S. Programming with idioms in APL. In *Proceedings of APL International Conference,* (Rochester, NY), 1979, 232-235.

[18] Prieto-Díaz, R. and Freeman, P. Classifying software for reusability. *IEEE Software, 4,* 1, (Jan. 1987), 94-104.

[19] Raymond, D. R. and Tompa, F. W. Hypertext and the *Oxford English Dictionary. Communications of the ACM, 31,* 7, (Jul. 1988), 871-879.

[20] Reasoning Systems Inc. *Refine User's Guide.* (Palo Alto, California), 1990.

[21] Rich, C. and Waters, R. C. *The Programmer's Apprentice.* Addison Wesley, 1990.

[22] Roget, P. *Roget's Thesaurus of English Words and Phrases.* George Davidson (Editor), Penguin Books, 2002.

[23] Rugaber, S., Ornburn, S. B. and LeBlanc, R. J. Jr. Recognizing design decisions in programs. *IEEE Software, 7,* 1, (Jan. 1990), 46-54.

[24] Shaw, M. Prospects for an engineering discipline of software. *IEEE Software, 7,* 6 (Nov. 1990), 15-24.

[25] Shaw, M. and Garlan, D. *Software Architecture / Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[26] Snetling, G. Concept analysis - A new framework for program understanding. In *SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'98),* (Montreal, Canada), 1998.

[27] Warmer, J. and Kleppe, A. *The Object Constraint Language, Second Edition.* Addison Wesley, 2003.

[28] *Webster's Seventh New Collegiate Dictionary.* G & C Merriam Co., 1965.

[29] Wills, L. M. *Automated Program Recognition by Graph Parsing.* Ph.D. Thesis, Massachusetts Institute of Technology, Technical report 1358, MIT Artificial Intelligence Laboratory, July 1992.

[30] Winchester, S. *The Professor and the Madman: A Tale of Murder, Insanity, and the Making of the Oxford English Dictionary.* G. K. Hall & Co., Thorndike, Maine, 1998.

[31] Yokoi, T. The EDR electronic dictionary. *Communications of the ACM, 38,* 11 (Nov. 1995), 42-44.