Status Report: Software Reusability

Rubén Prieto-Díaz
Reuse, Inc.
Fairfax, VA 22033

Introduction

Software reuse is the use of existing software artifacts during the construction of new software systems. The focus of reuse is not only source code fragments but all the intermediate workproducts generated during software development such as requirements documents, system specifications, design structures, and *any information the developer may need in the process of creating new software* [Free83].

The notion of reusability is an old idea that has been around since human beings became involved in problem solving. As solutions to current problems are found, these solutions are tried in similar new problems. Only certain elements of the solutions are successfully applied to new problems. Old solutions are typically modified, combined, and adapted to solve the new problem. As a proven solution is used over and over to solve the same type of problem, it becomes accepted, generalized, and standardized.

Formal generic mathematical models, for example, can be applied to solve specific problems across several engineering fields or domains. The successful abstraction of a class of problems in the mathematical domain, its solution in that domain, and the ability to apply that solution to specific instances in other domains not only captures the essence of the reuse problem but shows the tremendous potential of reuse at high levels of abstraction.

Reuse is also practiced at the artifact level. Elements used in the construction of artifacts can also be standardized and generalized for reuse. The movable typeset is a classical example of reuse of components that had a high impact on civilization. Examples of artifact reuse include: standard beams in civil engineering; bolts, nuts, and gears in mechanical engineering; and resistors, capacitors, and ICs in electronics. In software engineering, generic algorithms are the equivalent of mathematical solutions in other fields. Subroutines, macros, functions, and objects are examples of reusable software artifacts.

The problem in software engineering is not lack of reuse but lack of wide spread systematic reuse. Software reuse has been practiced informally since programming was invented. Programmers have been reusing sections of code from previous programs, subroutines from functional collections, and algorithms. They also adapt and reverse-engineer existing systems. All this, however, is done informally and very much ad-hoc. The need for a change is clearly stated by Neighbors:

> Our wish to build software systems from reusable software components represents a shift from craftsman production to mass-production. This shift is forced upon us by the ever increasing size of software systems we build [Neig92].

What has been the focus of the reuse research community is formalizing software reuse. Substantial quality and productivity pay-off from reuse are only achieved if conducted systematically and formally. The history of software reuse is characterized by a struggle towards formalization in an environment where pragmatic problems are the norm and their quick informal solutions usually take precedence.

Historical Background

The concept of formal software reuse was introduced by Dough McIlroy in what has become the seminal paper in software reusability [McIl68]. He introduced the notion of software reuse by proposing an industry of off-the-shelf standard source code components. He envisioned the construction of complex software systems from small building blocks available through catalogs.

His idea was key for the development of the software factory concept. Systems Development Corp. (SDC) was first in registering the concept of a "software factory" as a trademark in 1974 [BC75]. Although the SDC software factory established a well-defined set of procedures for a consistent and repeatable software process, their emphasis on reuse was implicit at best [Cusu91]. Reuse was not defined in the process but programmers were expected to reuse whatever they could, from code segments to previous experiences. Reuse was informal and ad-hoc. The SDC experience was used as basis for future software factories, especially in Japan, where formal reuse processes were later defined and integrated.

Also during the middle 70's Lanergan and Poynton began a reuse project at the Raytheon Missile Division. Their work was reported in another classic paper in 1979 [LP79] and is considered the first case of formalizing reuse at the organization level.

Interest in software reuse research spread through academia in the late 70's. Several research projects were initiated, the first was from the University of California, Irvine. [Free76]. Results of the research from this period is best exemplified in the proceedings of the First International Workshop on Reusability in Programming [wrp83]. A notable contribution of this workshop, besides the classical papers included in the proceedings, was the consensus that reuse should be pursued at several levels of abstraction, and the recognition that reuse is not only a technical problem but it includes several non-technical issues of significant relevance.

Numerous workshops, symposia, and seminars have followed since then and the number of publications in software reuse has exploded from only a few dozen in 1980 to several hundreds in 1990.

The creation of large-scale reuse programs has also contributed significantly to the field. The STARS Initiative [Mart83] in the U.S., EUREKA's Software Factory and many ESPRIT projects in Europe [ieee89] demonstrate not only the interest but the commitment of governments to promote software reuse. The corporate world is equally interested and committed as demonstrated by the software factories in Japan (Hitachi, Toshiba, NEC, Fujitsu, NT&T) and the several corporate reuse programs being established in the U.S. (GTE, AT&T, IBM, HP)

Reuse work intensified significantly in the late 80's. Several advances were made in library systems, classification techniques, creation and distribution of reusable components, reuse support environments, and corporate reuse programs. In spite of this effort, a common complaint was voiced consistently at workshops and conferences: reuse was not delivering its promise of significant increase in productivity and quality.

In 1988 Vic Basili advanced a broader definition of reuse, the "use of everything associated with a software project including knowledge." [BR88]. This new perspective has opened research doors in other disciplines and has contributed to gain further recognition of the ubiquity of the reuse problem. More recent works have addressed the previously neglected non-technical factors. These include managerial, economic, social, cultural and legal factors. It has been recognized that these non-technical problems are as important, and maybe more difficult to solve, than the technical problems. The present view of reuse attempts to integrate all these factors into the idea of "institutionalized" reuse.

A Reuse Taxonomy

A taxonomy of software reusability can be described by means of a faceted classification scheme. There are at least six dimensions or perspectives called facets that we can use to view software reusability. These facets are orthogonal in the conceptual plane and provide somewhat independent views. We can use these independent views in this status report to assess the state of the research, art, and practice, and to analyze the problems and issues in software reusability.

Software Reusability

{by-substance}	**{by-scope}**	**{by-mode}**	**{by-technique}**	**{by-intention}**	**{by-product}**
ideas, concepts	vertical	planned, systematic	compositional	as-is, black box	source code
artifacts, components	horizontal	ad-hoc, opportunistic	generative	with-modification, white box	design
procedures, skills					specifications
					objects
					text
					architectures

The *by-substance* facet defines the essence of the items to be reused; what is that we are reusing. The *by-scope* facet defines the form and extent of the reuse activity, the *by-mode* facet defines how the reuse activity is conducted, the *by-technique* facet defines what approach is used to implement reuse, the *by-intention* facet defines how reusable elements will be reused, and the *by-product* facet defines what are the software workproducts that are reused.

Reuse of Ideas— deals with reuse of formalized concepts such as general solutions to a class of problems. Reuse of formalized general solutions in software engineering is best exemplified by generic algorithms. The work of D. Knuth and the collection of algorithms from the ACM are good examples.

The state of the research, art , and practice in algorithm development has reached a relative mature stage. From the reuse perspective, however, there is a need to develop and distribute standard catalogs and to complement basic catalog information with specific and detailed information on how to reuse the algorithms.

Reuse of Artifacts— In software engineering the focus has been on source code components. Progress in this area is demonstrated by the software factories reported above and by several other corporate programs aimed at repeating the Raytheon experience. Reusable software parts collections such as the Booch Ada Parts and the EVB collection are good examples. Initial experimentation with parts technology is attributed to the CAMP (Common Ada Missile Package) project begun in the early 80's. Current OO techniques provide the most promising approach for component technology. Environments like Eiffel and Motif offer integrated object libraries for reuse.

Parts reuse research is now concentrating on parts quality including certification and reliability. Component adaptability and modification effort have also become important issues in research and in practice. Domain specific collections such as NASA's COSMIC and the USAAF's CARDS are emerging and promise to be the new trend in parts reuse.

Reuse of Procedures— is definitely one of the most intense areas of research. Software process programming and process-centered environments research focus on formalizing and encapsulating procedures for software development. The reuse community is looking into the possibility of creating collections of reusable processes that can be connected to instantiate new and more complex processes. Process reuse is one of the main tasks of DARPA's STARS Initiative. The STARS Conceptual Framework for Reuse Processes (CFRP) describes a set of reusable processes that can be combined to implement customized reuse programs in an organization.

Reuse of procedures also include skills; the reuse of "how-to" knowledge. This area has received significant attention from the expert systems community, but very little from the reuse community. Human skills are reused informally by reassigning personnel but no formal effort is made to capture and encapsulate such knowledge for others to reuse. This is an area where research is needed and one that offers significant potential for reuse. Domain analysis is beginning to explore methods and techniques for reuse of expert procedural knowledge.

Vertical Reuse— is the reuse of software within same domain or application area. The goal of vertical reuse is to derive generic models for families of systems that can be used as standard templates for assembling new systems. The narrower the domain the higher the pay-off in vertical reuse.

The focus of research in vertical reuse has been on domain analysis and domain modeling. Both approaches are concerned with the identification and development of domain models. Several domain analysis methods have been developed in the last three years. Examples include: FODA (Feature Oriented Domain Analysis) from the SEI, the Sandwich approach developed by Prieto-Díaz, the Synthesis method from the SPC (Software Productivity Consortium), and more recently, SofTech's Domain Analysis Guidelines. Also several OO methods have been extended to cover domain analysis and model development.

Although vertical reuse has been practiced informally in most software organizations, there is a definite trend towards a more formal and systematic practice. Software factories, for the most part, are aimed at vertical reuse, and organizations with large long-lived projects, such as NASA and the DoD services, are concentrating on vertical reuse. Specialized packages like Motif are examples of successful vertical reuse.

<u>Horizontal Reuse</u>— is the reuse of software across domains. The goal of horizontal reuse is the development of generic parts that can be used in different applications. Scientific subroutine libraries, the Unix tools, and the Booch Ada parts are good examples of horizontal reuse.

Although horizontal reuse research has been concerned with packaging and presentation of parts, its current focus has been on development of broad access libraries and library networks. Projects like ASSET (Asset Source for Software Engineering Technology) and DISA/CIM's Center for Software Reuse are examples of initiatives toward broad access repositories of common parts. Significant work is being done by these projects towards cataloging and classification standards and in the area of interoperability and networking of repositories. As these initiatives develop we will see an emerging industry of software parts covering both, horizontal and vertical reuse.

<u>Planned Reuse</u>— is the systematic and formal practice of reuse. It implies that guidelines and procedures for reuse have been defined, and that metrics are collected to assess and measure reuse performance. Planned reuse requires substantial up-front investment and commitment, but returns on such investments are difficult to predict. It requires a significant change in the current practice of software development and demands certain discipline and compromise from software practitioners. Planned reuse is the norm in software factories.

The recent "Reuse Adoption Guidelines" from the SPC are a good example of the state of the art in planned reuse. The SPC guidebook prescribes a step by step process to establish a reuse program. Reuse maturity models are at the core of planned reuse. They define a sequence of levels that organizations can progress through to achieve formal systematic reuse. Current reuse maturity models are inspired on the SEI's Process Capability Model. The SPC, for example, has developed a five-level maturity model: Ad-hoc, repeatable, portable, architectural, and systematic. The spur of research in this topic has produced several reuse maturity models.

A concern in planned reuse is the lack of economic models. Managers need to know expected benefit, return on investment, initial cost, and performance criteria before committing to a full-scale reuse program. Although some economic models have been proposed, they have not been validated. More research and experimentation is definitely needed in economic models of reuse.

Ad-hoc Reuse— is the informal practice of reuse. It is usually referred to as opportunistic reuse where components are selected from general reuse libraries. Typically, reuse libraries are populated with components that have not been designed for reuse, procedures for reuse have not been established, and reuse is conducted at the individual rather than at the project level.

Ad-hoc reuse has been very much the state of the practice in the industry. The emphasis in ad-hoc reuse has been the development of better and more capable reuse libraries with friendly interfaces and powerful retrieval mechanisms. The bottleneck, however, has been library population; cataloging and classification of reusable components remains a time consuming manual task. Although reuse libraries technology has experienced significant progress, research is needed to solve this bottleneck.

Examples of operational library systems include SofTech's RAPID, being used by DISA/CIM, and IBM's RLS, being used by ASSET. Several new reuse library systems, like SPS's InQuisiX, offer advanced features for browsing and retrieval, and promise significant improvement in overall reuse. Support for automatic cataloging and classification is still needed.

Compositional Reuse— is the use of existing software components as building blocks for developing new software systems. Compositional reuse technology is based on well-established collections, efficient library systems, and standard interfaces. Although it advocates reuse of all software workproducts, in practice, compositional reuse has focused mainly on source code reuse.

Research topics in compositional reuse include component selection and retrieval, component understanding, component adaptation, and component integration. Library systems address the first two, understanding and adaptation are related to domain analysis, and integration is an interface issue. Integrated environments are seen more and more as an effective approach to address these issues.

The Unix shell is a typical example of the state of the practice, while more recent OO environments, like Eiffel and Motiff that offer integrated object libraries for reuse, reflect more the state of the art. Prototype environments like ESPRIT's REBOOT (REuse Based on Object-Oriented Techniques) are examples of the trend towards formal compositional reuse. REBOOT includes associated methods to support software reuse by means of object-oriented technology. Other prototype systems include Paramax's RLF and CTA's KAPTUR; both use libraries based on knowledge representation models including semantic nets.

The state of the research in compositional reuse is exemplified by Batory's GENESIS system, a DBMSs generator. GENESIS is based on a well-defined conceptual framework of building-blocks for DBMSs. Interfaces, module semantics, and possible module interconnections are standardized. An open-ended library and a compiler tool are used to construct target DBMSs as composition of modules. Research focus is towards extending and generalizing the GENESIS approach. Domain analysis is seen as a key element of this research.

Generative Reuse— is reuse at the specifications level by means of applications or code generators. This concept of reuse offers the highest pay-off potential, but at the same time it is a difficult technology to scale up to industrial production. Systems like Refine and MetaTool demonstrate the potential of this approach.

Research in generative reuse has focused on formal representations of domain specific specification languages, software processes, and meta-generators. Domain analysis is becoming an important element for deriving vocabularies, architectures, and grammars for specific domains. The trend is towards meta-generators of domain specific application generators. This ultimate goal would make reuse a natural part of the process.

The state of the practice is domain specific application generators, many available commercially, while the state of the art is systems like MetaTool and Refine. Perhaps the most well-known tools for building generators are Lex and YACC which are themselves generators to build lexers and parsers. The state of the research has focused on extension of these approaches to broader domains and to more powerful tools. Neighbor's Draco system is a typical example of application-generator generators beyond lexer and parser generators. Draco uses domain analysis to develop a formal domain specific grammar and a set of basic source-to-source transformations. The grammar and transformations are then used with parser generators, pretty-printers, and other generator tools to produce executable code from high-level specifications. There is significant research interest to extend the Draco paradigm.

Black-box Reuse— is the reuse of software components without any modification. Black-box reuse is synonymous with reuse "as-is". Typically, reusable software components are packaged and their interactions defined by means of standard interfaces. Although this approach to reuse guarantees higher quality and reliability of software systems, creation of reusable black boxes is more costly and involved.

The Ada language and to some extent OO programming lend more naturally to black box reuse. Concepts of information hiding and inheritance are key for creating highly modular and independent components. One of the key issues in black box reuse is verification and certification of reusable components; how to demonstrate that a given component will perform flawless under all possible conditions. This problem has drawn significant research interest especially in the area of formal specifications. Formal specifications can be used to prove programs correct, and therefore reduce exhaustive testing.

Although black-box reuse is being heralded as the approach to create truly reliable systems at low cost, the first collection of certified components is yet to appear. The STARS initiative and several DoD services are engaged in the creation of domain specific certified components.

<u>White-box Reuse</u>— is the reuse of components by modification and adaptation. This is by far the most common approach, mainly when reuse is conducted informally. The trend in white box reuse is towards parameterization and built-in adaptability. Domain analysis, again, plays a key role in identifying variability in families of systems. More research in domain analysis methods is needed to improve white box reuse.

Most of the existing reuse programs, including software factories, use white box reuse. As reuse programs grow and evolve, reuse by adaptation is becoming more formalized. C libraries and the Unix shell are examples of flexible components. The trend in white box reuse is to expand reuse to higher level products like designs and specifications.

<u>Reuse Products</u>— The by-products facet in the classification scheme lists the kinds of software workproducts that can be reused. This is only a partial but representative list of the state of the practice.

Source code is the workproduct that has received the most attention. Reuse practice focus mainly on reuse of code components. Most reuse tools, environments, and methods are aimed at the reuse of code. Although the trend is towards higher level reuse, the state of the art and the state of the practice remains at the code level. As other aspects of reuse evolve, we will see less and less emphasis on code reuse. Code will eventually be generated automatically from higher level representations and systems developers will not have to deal with source code any more than they deal with assembly code now.

Designs offer higher pay-off in reuse than source code. Reuse of designs is still an emerging technique. Designs can be partially or indirectly reused through OO design methods, but a systematic practice is still at the research stage. Examples of research prototypes that support reuse at the design level include MCC's IDeA environment. IDeA supports reuse of abstract software designs represented in the form of design schemas. The system uses software specifications to find matching designs from a design reuse library.

Specifications include the reuse of designs and code. When a specification is made available for reuse it is typically bundled with its respective implementations at the design and code levels. Specifications reuse is the focus of generative reuse, as discussed above, and offers the highest pay-off. The states of the practice, art, and research are the same as generative reuse.

Objects are gaining ground as the most reusable workproducts. There are several methods, tools, and complete environments for creating programming objects. The trend in OO is towards integration of tools and methods, and coverage of all software development stages. OO techniques like Shlaer/Mellor, Rumbaugh, and Booch cover from domain analysis to object programming and also provide support tools. OO is seen as the technique of the future for reuse. It can be used to support, at least partially, planned, vertical, and compositional reuse.

Text is also a reusable product. Since all development products, except code, consist of documents intended for human consumption, reuse of text is becoming increasingly important. This aspect of reuse is still at the research level, but the wealth of knowledge and techniques available from information retrieval promises a quick development. The ESPRIT's Practitioner project, for example, is developing tools to index, catalog, and manipulate units of text for reuse. Hypertext is also a key technology for reusing text. The trend in this aspect of reuse is towards integration of reusable text with all other workproducts to increase overall reuse.

Architectures can be seen as the largest unit of reuse. Reuse of architectures is the goal of DARPA's DSSA Project. Applications domains are analyzed to find generic designs that are then used as basic templates for integrating reusable parts or for developing specialized code generators. This is still a research area that complements and supports both, generative and compositional reuse.

The Future

Software reuse has made significant progress in the last 25 years but it is still an elusive technology. The concepts are simple yet applying them systematically remains a challenge. Three distinctive trends have emerged in the last few years that show promise of finally reaching closure on the reuse problem: 1- domain analysis and domain engineering provide the means and support for implementing pre-planned reuse; 2- software reuse processes help integrate reuse into software development; and 3- megaprogramming, which is poised to establish the new paradigm of software development: domain-specific, reuse-based, and process-driven.

Domain Analysis

Domain analysis was introduced by Jim Neighbors in 1980 [Neig80] as the activity of "identifying objects and operations of a class of similar systems in a particular problem domain." Several domain analysis methods have been proposed in the last few years and there is significant interest to automate the process at least partially. An emergence of domain analysis support tools is imminent in the upcoming years.

Domain analysis holds the key for a systematic, formal, and effective practice of software reuse. It is through domain analysis that domain knowledge is transformed into generic specifications, designs, and architectures for reuse in developing new software systems within the domain. Availability of generic templates provides the basis for creating truly reusable components and for ease of reuse by composition.

Reuse and the Software Process

Another key factor for successful reuse is our ability to make it part of the software development process. The recent interest in characterizing reuse using maturity models and adoption processes is a clear sign of progress. Reuse maturity models provide the needed framework for tools and methods' development while adoption processes define methods and procedures for integrating reuse as standard practice in developing software. Successful integration of guidelines for reuse practice will definitely make a difference in the process towards institutionalization of reuse practice. The software industry is eager to adopt reuse if only they are told how.

Megaprogramming

The idea of megaprogramming (MP) was introduced by Bohem and Sherlis at the DARPA Software Technology Conference in April of 1992 [BS92]. MP refers to the "Practice of building software by components in a context of architecture conventionalization and reuse." Although programming within a megamodule can be handled by traditional technology, computations spanning several megamodules are specified by megaprograms in a megaprogramming language.

MP is based on the ideas of domain engineering and is the main trust of the STARS Program. MP objective is to integrate current reuse technology such as reuse library tools, repositories, and reuse techniques into a systematic framework of industrialized reuse. Although MP bears resemblance to the original ideas of McIlroy, the conceptual and technological developments of the last 25 years place MP in a more realizable position.

Conclusion

Although software reuse has witnessed substantial progress in the last 25 years there are significant challenges ahead. It is through the continuous and persistent effort of researchers and industry that these challenges will be overcome. Recent awareness of the difficulty of the non-technical problems has begun to stir the community into addressing organizational, management, economic, cultural, social, and legal aspects of software reuse. The near future will see significant progress in these areas and hopefully witness the institutionalization of reuse. Reuse, in the end, should come so natural that we do not have to think about it.

References

[BC75] Bratman, H. and T. Court, "The Software Factory," *IEEE Computer*, **8**(5):28-37, May 1975.

[BR88] Basili, V.R. and H.D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment.* Tech. Report CS-TR-2158, Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742, December, 1988.

[BS92] Bohem, B and W.L. Scherlis, "Megaprogramming." In *Proceedings of the DARPA Software Technology Conference 1992*, pp 63-82, Los Angeles, CA, April 28-30, (Meridien Corp., Arlington, VA) 1992.

[Cusu91] Cusumano, M.A. *Japan's Software Factories,* Oxford University Press, New York, 1991.

[Free76] Freeman, P. *Reusable Software (Research Proposal)* University of California, Irvine, ICS Dept., 1976.

[Free83] Freeman, P. “Reusable Software Engineering:Concepts and Research Directions.” In Alan Perlis, editor, *Proceedings of the Workshop on Reusability in Programming*, pages 2-16, ITT Programming, Newport, RI, September, 1983.

[ieee89] *IEEE Software,* Special Issue on the ESPRIT Program, **6**(6), November, 1989.

[LP79] Lanergan, R.G. and B.A. Poynton "Reusable Code: The Application Development Technique of the Future." In *Proceedings of the IBM SHARE/GUIDE Software Symposium*, IBM, Monterey, CA, October, 1979.

[McIl68] McIlroy, M.D. "Mass-produced Software Components." In *Software Eng. Conepts and Techniques, 1968 NATO Conf. Software Eng.*, ed. J.M. Buxton, P. Naur, and B. Randell, pp 88-98, 1976.

[Mart83] Martin, E.W. “Strategy for a DoD Software Initiative,” *IEEE Computer*, **16**(3):52-59, March, 1983.

[Neig80] Neighbors, J. *Software Cons-truction Using Components.* Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1980.

[Neig92] Neighbors, J. “The Evolution from Software Components to Domain Analysis.” *International Journal of Software Engineering and Knowledge Engineering* **2**(3):325-354, September, 1992.

[wrp83] *Proceedings of the Workshop on Reusability in Programming*, Alan Perlis, editor, ITT Programming, Newport, RI, September, 1983.

SIDE BAR

• ASSET - Asset Source for Software Engineering Technology. Project sponsored by DARPA under STARS initiative. Building 2600, Suite 2, 2611 Cranberry Square, Morgantown, WV 26505, (304)594-3954

• Booch Method - Grady Booch*, Object Orieneted Design with Applications*, Benjamin/Cummings, Redwood CIty, CA 94065, 1991.

• Booch Ada Parts - Grady Booch, *Software Components with Ada: Structures, Tools, and Subsystems,* Benjamin/Commings, Redwood City, CA 94065, 1987. Source code of the components listed in this book is available from Grady Booch, Wizard Software, Inc., 2171 South Parfet Court, Lakewood, CO, (303)986-2405

• CARDS - Central Archive for Reusable Defense Software. Project sponsored by the U.S. Air Force Electronics System Center, 1401 Country Club Rd., Fairmont, WV 26554, (304)367-0421.

• Center for Software Reuse Operations - The CSRO is an element of the DoD Software Reuse Initiative under the Defense Information Systems Agency/Center for Information Management (DISA/CIM). 500 N. Washington St., Suite 101, Falls Church, VA 22046, (703)536-7485.

• CAMP - Common Ada Missile Packages, project sponsored by McDonnell Douglas Missile Systems Co. and the U.S. Air Force. USAF CAMP, AFATL/FXG, Elgin AFB, FL 32542-5434, (904)882-2961.

• COSMIC - COmputer Software Management and Information Center, a software catalog distributed by NASA. COSMIC, The University of Georgia, 382 East Broad St., Athens, GA 30602, (404)542-3265

• Domain Analysis Guidelines (DRAFT), DoD Software Reuse Initiative, DISA/CIM, 701 South Courthouse Rd., Arlington, VA 22204-2199, May, 1992, (703)285-6589.

• Donald Knuth, *The Art of Computer Programming,* Vol. 1 (Fundamental Algorithms), Vol. 2 (Seminumerical Algorithms), and Vol. 3 (Sorting and Searching), Addison Welsey, Reading, MA, 1968, 1969, 1973.

• Draco - Jim M. Neighbors, The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering* **SE-10**(5):564-573, September, 1984.

• DSSA- *The Domain-Specific Software Architecture Program*, LTC Erik Mettala and Marc H. Graham, Eds., Special Report CMU/SEI-92-SR-9, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, June, 1992.

• Eiffel - an Object Oriented language and environment available from Interactive Software Engineering, Inc., 270 Stroke Rd., Suite 7, Goleta, CA 93117, (805)685-1006.

• FODA - *Feature-Oriented Domain Analysis*, Kyo C. Kang, et.al., Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, November, 1990.

• Genesis - Don Batory, Concepts for a Database Compiler, *Proc. ACM Principles of Database Systems Conference*, 1988.

• GRACE - Generic Reusable Ada Components for Engineers. A library of 275 reusable software components available from EVB Software

Engineering, Inc. 5303 Spectrum Drive, Frederick, MD 21701, (301)695-6960.

• IDeA - Intelligent Design Assistant, Mitch Lubars, “Domain Analysis and Domain Engineering in IDeA.” In *Domain Analysis and Software Systems Modeling*, R. Prieto-Díaz and G. Arango (Eds.), IEEE Computer Society Press, 1991, pp: 163-178.

• InQuisiX - a classification and search system for software reuse available from Software Productivity Solutions, 122 Fourth Ave., Indialantic, FL 32903, (407)984-3370.

• KAPTUR - Knowledge Adquisition for Preservation of Trade-offs and Underlying Rationales. Sidney Bailin, *KAPTUR: A Tool for the Preservation and Use of Engineering Legacy*, CTA Inc. 6116 Executive Blvd., Suite 800, Rockville, MD 20852, (301)816-1200.

• Lex - a lexical analyzer generator tool available from the Unix System.

• MetaTool - an application-generator generator tool available from AT&T Bell Laboratories, 20 Shattuck Rd., Andover, MA 01810.

• Motif - a window manager environment that runs on top of X-windows. It is available from Open Software Foundations, 11 Cambridge Center, Cambridge, MA 02142, (617)621-8700

• Practitioner Project - ESPRIT Project P-1094.

• RAPID - Reusable Ada Packages for Information systems Development, a reuse library system used by DISA/CIM in its CSRO developed by SofTech, Inc., 460 Tottem Pond Road, Waltham, MA 02254.

• REBOOT - REuse Based on Object Oriented Techniques, a reuse-based software development environment developed as part of ESPRIT-2 and available from Bull S.A., Rue Jean Jaures, F-78340 Les-Clayes-Sous-Bois, France, (33-1)3080-7448.

• Refine - a transformations-based software development tool from Reasoning Systems, Inc., 3260 Hillview Ave., Palo Alto, CA 94304, (415)494-6201.

• Reuse Adoption Guidebook - Technical Report SPC-92051-CMC, Software Productivity Consortium, 2214 Rock Hill Rd., Herndon, VA 22070, November, 1992.

• RLF - Reuse Library Framework, a reuse library system used by CARDS and available from Paramax Systems Corporation, 12010 Sunrise Valley Drive, Dept. 7720, Reston, VA 22091, (703)620-7475.

• RLS - Reuse Library System, the reuse library system used by ASSET and developed by IBM and SAIC (Scientific Applications International Corporation).

• Rumbaugh method - J. Rumbaugh, et.al., *Object Oriented Modeling and Design*, Prentice Hall, Engewood Cliffs, NJ, 1991.

• Sandwish method - a domain analysis method developed by Reuse, Inc. and available in *Reuse Library Process Model*. IBM STARS Technical Report CDRL 03041-001, U.S. Air Force Electronic Systems Center, Hanscom Air Force Base, MA 01731, July, 1991.

• Shlaer/Mellor method - Sally Shlaer and Stephen J. Mellor, *Object Lifecycles: Modeling the World in States,* Yourdon Press, Englewood Cliffs, NJ 07632, 1992.

• STARS - Software Technology for Adaptable and Reliable Systems

• Synthesis - a software development approach developed by the SPC, Grady Campbell, et.al., Synthesis Guidebook, SPC-91122-MC, Software Productivity Consortium, 2214 Rock Hill Rd., Herndon, VA 22070, 1991.

• YACC - Yet Another Compiler-Compiler, a compiler generator tool available from the Unix System.