Software Engineering Institute

A Domain Analysis Bibliography

J. Hess
S. Cohen
B. Holibaugh
K. Kyang
A.S. Peterson
W. Novak
P. Carroll

June 1990

SPECIAL REPORT
CMU/SEI-90-SR-003

Domain Analysis Project

Carnegie Mellon

This technical report was prepared for the

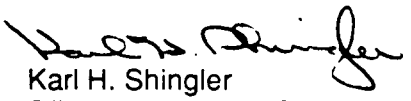
SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

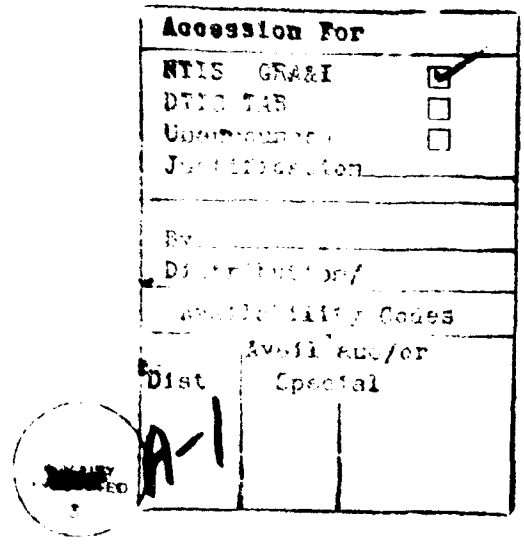
FOR THE COMMANDER



Karl H. Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents



1. Introduction

This document presents a bibliography of references on a comparatively new discipline called *domain analysis*. This discipline defines a process to identify and represent the relevant information in a *domain* (a set of systems which share common capabilities). The information is derived from:

1. the study of existing systems and their development histories
2. knowledge captured from domain experts
3. underlying theory
4. emerging technology

Domain analysis has received considerable attention since the early 1980s. This interest stems from the fact that the application of domain analysis is now believed to be part of the foundation upon which a successful and systematic program of software reuse can be built. This foundation is achieved by capturing and preserving the information and expertise associated with an application domain. Domain analysis allows this information to be reused in future developments in the form of application-specific tools and reusable software models, architectures, and components.

This bibliography has been compiled as a part of the work on the Domain Analysis Project at the Software Engineering Institute. The bibliography's purpose is to provide an historical perspective on the field as well as a necessary background for further work in the discipline.

In addition to the appropriate publication information for each document type, all citations include either 1) an *abstract* taken from the document itself, usually written by the author, or 2) an *annotation* written for the document by the compilers of this bibliography. In most cases the author's abstract was used. These abstracts and annotations should help the reader to determine if there is interest in a given citation. Various indices are also included to simplify the task of locating a particular reference or a range of articles in a subject area.

1.1. Selection Criteria

The amount of available literature describing software reuse in general, and domain analysis in particular, has increased dramatically over the past 7 to 10 years and continues to grow. Due to this growth it is impossible to call any bibliography "complete" by the time it is published; in fact, it is unlikely to include every relevant publication while the bibliography is being researched. The authors applied the following questions to each potential reference:

1. Does the reference *explicitly* discuss an aspect of domain analysis, such as domain engineering, domain knowledge or expertise, domain modelling, domain-specific architectures, domain analysis methodologies, etc.?
2. Is the reference very closely related to an aspect of the domain analysis process, especially one related to software reuse, although it might not refer to it as such?
3. Is the reference a seminal historical reference upon which later work in domain analysis was based?
4. Does the reference *not* belong to another well-defined field of research outside of domain analysis?

It is important to determine if a reference primarily belongs to a field of the literature separate from domain analysis. This is pertinent to many articles, such as those concerning the formal specification of

software systems, or knowledge acquisition/representation for expert systems. This is a gray area of overlap, as both formal specification and the representation of knowledge are relevant to domain analysis. [1] Some of the most significant articles in these related areas are included in the bibliography. These considerations helped to pare down a large collection of tangentially relevant material to a more useful set of references that are directly applicable to domain analysis.

Finally, some otherwise relevant citations were excluded from the bibliography if they were

1. proprietary to an organization,
2. copies of slide presentations (which are typically unpublished and difficult to understand without the accompanying talk),
3. difficult to obtain, or
4. superseded by later, better defined work of the same author(s).

If an appropriate publication has been omitted from this bibliography please contact the authors at:

Domain Analysis Project
Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

1.2. Using the Bibliography

This document contains the following sections:

1. The set of full document citations (Chapter 2)
2. A table showing the major subject classifications of the references (Chapter 3)
3. An alphabetical cross reference by author's name (Appendix I)
4. A chronological ordering by year of publication (Appendix II)
5. An alphabetical cross reference by project and sponsor names (Appendix III)
6. An alphabetical index listing the citation key and the page number where the full citation may be found (Index)

The classification table in Chapter 3 divides the citations into six subject categories:

1. *information gathering*
2. *domain analysis methodology*
3. *tools and environment support*
4. *representation (domain models and software architectures)*
5. *application domains*
6. *management issues*

Within this bibliography the citation key (such as PRIE89C) is used as a unique identifier of the document. If such a key is found in (for example) the classification table, that same key can be referenced

[1] In fact, the differences between knowledge acquisition for use in domain analysis and an expert system are more in the *use* of the knowledge than in the analysis process itself.

in the index to find the page number for the full text citation.

For those readers who are unfamiliar with domain analysis, the following references may provide a suitable starting point:

- GILR89A: Impact of Domain Analysis on Reuse Methods
- KANG89A: Results from Domain Analysis Working Group
- PRIE89A: Domain Analysis Tutorial

The following references may be of interest to those readers who require information on different methodologies that may be applied to the domain analysis process:

- GILR89A: Impact of Domain Analysis on Reuse Methods
- PRIE87C: Domain Analysis for Reusability

2. Bibliography

[ADEL85A] Beth Adelson & Elliot Soloway.
The Role of Domain Experience in Software Design.
IEEE Transactions on Software Engineering, vol. SE-11, no. 11: pp. 1351-1360, November, 1985.
IEEE. Reprinted with permission.

Annotation: In the experiment described in this article three expert and two novice designers were presented with both familiar and unfamiliar design problems. The designers were given three different types of systems. The designers might or might not have had previous experience with the domain of the object or the actual object. The designers' behavior indicated how tools should be designed to work for people of varying domain knowledge. Behavior was monitored and the following results observed: *Behavior:* (1) Forming mental models to simulate the design in progress, (2) Simulating the model to integrate familiar materials in novel ways and check behavior of the model, (3) Systematic expansion to allow for smooth functioning of the simulation, (4) Making implicit constraints explicit so as to allow for simulation, (5) Labeling plans so previous solutions could be reused, and (6) Note-taking to allow for systematic expansion without overlooking important concerns not at the current level of detail. *Factors affecting behavior:* (1) Simulation and note-taking happened when the designer had a certain level of prior knowledge of the domain, (2) Designers with inadequate domain knowledge were quick to constrain their designs so as to get a sufficiently specific model so the simulation could go ahead, and (3) Designers who had previously planned in their assigned domain used existing plans rather than formulating constraints, simulating and taking notes.

[ALEX86A] James H. Alexander, et al.
Knowledge Level Engineering: Ontological Analysis.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Pages 963-968. AAAI, August, 1986.
American Association for Artificial Intelligence. Reprinted with permission.

Abstract: Knowledge engineering suffers from a lack of formal tools for understanding domains of interest. Current practice relies on an intuitive, informal approach for collecting expert knowledge and formulating it into a representation scheme adequate for symbolic processing. Implicit in this process, the knowledge engineer formulates a model of the domain, and creates formal data structures (knowledge base) and procedures (inference engine) to solve the task at hand. Newell (1982) has proposed that there should be a *knowledge level* analysis to aid the development of AI systems in general and knowledge-based expert systems in particular. This paper describes a methodology, called ontological analysis, which provides this level of analysis. The methodology consists of an analysis tool and its principles of use that result in a formal specification of the knowledge elements in a task domain.

[ALLE87A] Bradley P. Allen & Peter L. Holtzman.
Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project.
In *Proceedings of the Space Operations Automation and Robotics Workshop*, Pages 407-410. NASA, Johnson Space Center Houston, TX, August, 1987.

Abstract: We provide an overview of the Automated Software Development Workstation Project, an effort to explore knowledge-based approaches to increasing software productivity. The project focuses on applying the concept of domain-specific automatic programming systems (D-SAPSs) to application domains at NASA's Johnson Space Center. We describe a

version of a D-SAPS developed in the Phase 1 of the project for the domain of Space Station momentum management, and discuss how problems encountered during its implementation have led us to concentrate our efforts on simplifying the process of building and extending such systems. We propose to do this by attacking three observed bottlenecks in the D-SAPS development process through the increased automation of the acquisition of programming knowledge and the use of an object oriented development methodology at all stages of program design. We discuss how these ideas are being implemented in the Bauhaus, a prototype CASE workstation for D-SAPS development.

[ARAN88A] Guillermo F. Arango.
Domain Engineering for Software Reuse.
PhD thesis, University of California at Irvine, 1988.
Abstract:: A precondition for software reuse is the existence of reusable information. There is a lack of systematic methods for producing reusable information. This dissertation proposes a method for practical *domain analysis*, defined as the process of identification, acquisition, and evolution of information to be reused in the construction of software systems for restricted classes of applications, or problem domains. The method for domain analysis is presented in the context of a *domain engineering framework*. A framework is not a theory and the paper does not offer a detailed canonical scheme of how every type of domain analysis is or ought to be done.

[ARAN88B] Guillermo F. Arango.
Evaluation of a Reuse-Based Software Construction Technology.
In *Proceedings of the Second IEE/BCS Conference: Software Engineering 88*, Pages 85-92. IEE, London, UK, July, 1988.
Annotation: The author has been involved with Draco, a technology for software construction based on the reuse of software components. He shares some of the lessons learned from the development and evaluation of Draco, and discusses the features that condition the transfer and adoption of domain-based technologies. He presents an historical account of the evolution of the Draco technology and distinguishes the paradigm for reuse-based software construction from the technological assembly of the Draco system.

[ARAN88C] Guillermo F. Arango & Eiichi Teratsuji.
Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse.
Technical report ASE-RTP-87, ICS, University of California, Irvine, CA, July, 1988.
Abstract: This report illustrates aspects of the application of a clustering function to uncover regularities among classes of applications in restricted problem domains. Clustering is used as a means to identify reusable patterns of first-order information. Patterns are represented in terms of classification trees which are used for two purposes, (1) for identifying 'packages' containing reusable patterns, and (2) to drive the elicitation of information from experts. The motivation for choice of this technique and the context for its application is discussed in [ARAN88A]. The next two sections provide some context. This report is *not* intended to be self-contained, but is one component in a collection.

[ARAN89A] Guillermo F. Arango.
Domain Analysis - From Art Form to Engineering Discipline.
In *Proceedings of the Fifth International Workshop on Software Specification and Design*. Pages 152-159. IEEE Computer Society, Washington, DC, May, 1989.
Abstract: A precondition for reusability in software development is the existence of reusable resources. There is a lack of systematic methods for producing

reusable information. Within the 'reuse community', there is belief that *domain analysis* will facilitate the identification and capture of reusable abstractions for restricted classes of applications. This belief is grounded on successful experiences conducted by some very capable individuals. However, for domain analysis to become a practical technology we need: (1) to understand the conceptual foundations of the process; (2) to produce an unambiguous definition in the form of specific techniques; and, (3) to provide adequate support tools. In this paper we advance a conceptual framework. We do not offer a detailed, canonical, scheme of how *every* type of domain analysis is or ought to be done. We have identified a set of principles providing coherence to a diverse set of findings about domain analysis. Within this framework we have explored techniques for *practical* domain analysis. The framework is useful for comparing between different approaches to domain analysis, and can be used as guidance in developing other instances of methods and representations.

[ASDJ88A] Maryam Asdjodi.
Knowledge-Based Component Composition: An Approach to Software Reusability.
PhD thesis, University of Alabama at Huntsville, 1988.
Annotation: This research focused on the component composition approach as a high-level method for software generation. A prototype system was designed for semi-automatic generation of software for an application domain, using the knowledge of the domain and a library of objects, functions, and templates. The components of the prototype system include user interface, library, library interface, text generator, user's environment table, object transformation base, and a pool of instantiators.

[BAIL88B] Sidney C. Bailin.
Semi-Automatic Development of Payload Operations Control Center Software.
Report prepared for NASA Goddard Space Flight Center , Computer Technology Associates, Laurel, MD, October, 1988.
Abstract: This report summarizes the results of a domain analysis effort of Payload Operations Control Center (POCC) software done for NASA Goddard Space Flight Center. It presents the results of the domain analysis, and proposes an approach to semi-automatic development of POCC Application Processor (AP) software based on these results. The domain analysis enabled us to abstract, from specific systems, the typical components of a POCC AP. We were also able to identify patterns in the way one AP might be different from another. These two perspectives -- aspects that tend to change from AP to AP, and aspects that tend to remain the same -- suggest an overall approach to the reuse of POCC AP software. We found that different parts of an AP require different development technologies. We propose a hybrid approach that combines *constructive* and *generative* technologies. Constructive methods provide for automated generation of software from specifications in a very high-level language (VHLL). In the next phase of our effort we propose to demonstrate how these technologies can be combined to facilitate AP development.

[BAIL89C] Sidney C. Bailin.
Generic POCC Architectures.
Report prepared for NASA Goddard Space Flight Center , Computer Technology Associates, Laurel, MD, April, 1989.
Abstract: This document describes a generic Payload Operations Control Center (POCC) architecture based upon current POCC software practice, and several refinements to the architecture based upon object oriented design principles and expected developments in teleoperations. The current technology generic architecture is an abstraction based upon close analysis of the ERBS, COBE, and GRO POCCs. A series of three refinements is

presented: these may be viewed as an approach to a phased transition to the recommended architecture. The third refinement constitutes the recommended architecture, which, together with associated rationales, will form the basis of the rapid synthesis environment to be developed in the remainder of this task. The document is organized into two parts. The first part describes the current generic architecture using several graphical as well as tabular representations or 'views'. The second part presents an analysis of the generic architecture in terms of object oriented principles. On the basis of this discussion, refinements to the generic architecture are presented, again using a combination of graphical and tabular representations.

[BAIL89D] Sidney C. Bailin.
The KAPTUR Environment: An Operations Concept.
Report prepared for NASA Goddard Space Flight Center , Computer Technology Associates, Laurel, MD, June, 1989.
Annotation: This report presents a high-level specification and operations concept for *KAPTUR*- a development environment based on Knowledge Acquisition for Preservation of Trade-offs and Underlying Rationales. KAPTUR is intended to do what its name implies: to capture knowledge that is required or generated during the development process, but that is often lost because it is *contextual* (i.e., it does not appear directly in the end-products of development). Such knowledge includes issues that were raised during development, alternatives that were considered, and the reasons for choosing one alternative over others. Contextual information is usually only maintained as a memory in a developer's mind. As time passes, the memories become more vague and individuals become unavailable, and eventually the knowledge is lost. KAPTUR seeks to mitigate this process of attrition by recording and organizing contextual knowledge as it is generated. KAPTUR also seeks to facilitate the application of knowledge to future developments. From the relations between past and ongoing work, developers can fortify their understanding of the current problem and its possible solutions.

[BARS85A] David R. Barstow.
Domain-Specific Automatic Programming.
IEEE Transactions on Software Engineering, vol. SE-11, no. 11: pp. 1321-1336, November, 1985.
IEEE. Reprinted with permission.
Abstract: Domain knowledge is crucial to an automatic programming system and the interaction between domain knowledge and programming at the current time. The PhiNIX project at Schlumberger-Doll research has been investigating this issue in the context of two application domains related to oil well logging. Based on these experiments, we have developed a framework for domain-specific automatic programming. Within the framework, programming is modeled in terms of two activities, formalization and implementation, each of which transforms descriptions of the program as it proceeds through intermediate states of development. The activities and transformations may be used to characterize the interaction of programming knowledge and domain knowledge in an automatic programming system.

[BATO88A] Don S. Batory.
Building Blocks of Database Management Systems.
Technical report TR-87-23, University of Texas, Austin, TX, February, 1988.
Abstract: We present a very simple formalism based on parameterized types and a rule-based algebra to survey and identify the storage structure and query processing algorithm building blocks of database management systems. We

demonstrate building block reusability by showing how different combinations of a few blocks yields the structures and algorithms of three different systems, namely System R (centralized), R* (distributed), and GRACE (database machine). We believe that codifying knowledge of DBMS implementations is an important step toward a technology that assembles DBMSs rapidly and cheaply from libraries of pre-written components.

[BATO88B] Don S. Batory.
Concepts for a Database System Compiler.
Technical report TR-88-01, University of Texas, Austin, TX, January, 1988.
Abstract: We propose a very simple formalism based on parameterized types and a rule-based algebra to explain the storage structures and algorithms of database management systems. Implementations of DBMSs are expressed as equations. If all functions referenced in the equations have been implemented, the software for a DBMS can be synthesized in minutes at little cost, in contrast to current methods where man-years of effort and hundreds of thousands of dollars are required. Our research aims to develop a DBMS counterpart to today's compiler-compiler technologies.

[BATO88C] Don S. Batory, J. R. Barnett, J. Roy, B. C. Twichell & Jorge F. Garza.
Construction of File Management Systems from Software Components.
Technical report TR-88-36, University of Texas, Austin, TX, October, 1988.
Abstract: We present an approach for developing building-block technologies for mature software domains. It relies on in-depth studies of existing systems, published algorithms and structures to discern generic architectures for large classes of systems. An architecture is a template in which building-blocks can be plugged. Interfaces are standardized to make blocks interchangeable. In this paper we describe our most recent prototype, a file management system (FMS) synthesizer. The synthesizer enables a customized FMS to be assembled from pre-written components in minutes at little cost. Writing the same FMS from scratch requires man-years of effort and hundreds of thousands of dollars.

[BENN84A] James S. Bennett.
ROGET: Acquiring the Conceptual Structure of a Diagnostic Expert System.
In *Proceedings of the IEEE Workshop on Principles of Knowledge Based Systems*, Pages 83-88. IEEE, Washington, DC, December, 1984.
Abstract: This paper describes ROGET, a knowledge-based system that assists a domain expert with an important design task encountered during the early phases of expert system construction. ROGET conducts a dialogue with the expert to acquire the expert system's conceptual structure, a representation of the kinds of domain-specific inferences that the consultant will perform and the facts that will support these inferences. ROGET guides this dialogue on the basis of a set of advice and evidence categories. These abstract categories are domain independent and can be employed to guide initial knowledge acquisition dialogues with experts for new applications. This paper discusses the nature of an expert system's conceptual structure and describes the organization and operation of the ROGET system that supports the acquisition of conceptual structures.

[BIGG88B] Ted J. Biggerstaff.
The Nature of Semi-Formal Information in Domain Models.
Technical report STP-289-88, Microelectronics and Computer Technology Corporation, Austin, TX, September, 1988.
Annotation: This article describes several levels of abstraction, from code to conceptual abstraction, in domain models. The lowest level is *code*, whose purpose is the execution of instructions. It is an implementation-specific abstraction that is constrained mostly by the programming language used. It

is formal object and exists in an operational form. The next abstraction is *software engineering design,* whose purpose is to abstract away detail. It is weakly related to informal concepts, and is also implementation specific. It is constrained by the language used and the application domain. It is considered to be a semi-formal object that is abstractly operational, and presents the system in reduced detail. The third abstraction is *generalized software engineering design,* whose purpose is also to abstract away detail. It is also weakly related to informal concepts, and is at a sufficiently high level to provide widely reusable designs. The last is *conceptual abstraction,* whose purpose is to strongly relate to informal concepts. Conceptual abstractions are not implementation specific, have an object-like structure, and assume a non-operational (prescriptive) form. The report also describes the use of informal knowledge in design recovery and briefly describes the DESIRE system they are constructing.

[BORG84A] Alexander Borgida, John Mylopoulos & Harry K. T. Wong .
Generalization/Specialization as a Basis for Software Specifications,
On Conceptual Modeling, Pages 87-117. Springer-Verlag, New York, NY, 1984.
Annotation: The paper describes a software specification methodology based on the notion of concept specialization. The methodology, which is useful for information systems, applies uniformly to the various components of these systems, such as data classes, inheritance, transactions, exceptions, and user interfaces (also called scripts). Its goal is the systematic and structured description of highly detailed world models, where concepts occur in many variations. An example from the domain of university information systems is used to illustrate and motivate the approach. The paper's key point is that generalization should be used as a cornerstone in designing data-intensive applications. Generalization hierarchies help the designer organize the process of gathering detail and integrating it into a coherent information system.

[BORG85A] Alexander Borgida.
Features of Languages for the Development of Information Systems at the Conceptual Level.
IEEE Software, vol. 2, no. 1: pp. 63-73, January, 1985.
Annotation: The fundamental observation underlying this article is that an information system can be viewed as a model of the real world. Information systems actually deal with the concepts that guide the way we think of the world. They are conceptual models. To express the information in a conceptual model, we need a language. Traditional languages are based on one of several data models which are more appropriate for modeling the way data are stored and accessed in the computer than the concepts underlying them. Because of this, there has been considerable research into semantic data models which allow conceptual models to be developed more conveniently. In conceptual model languages (CMLs), an object can exist without having a unique identifier and yet be distinct from other objects. This is in contrast with the relational model, where an object cannot be stored without a primary key. The author goes on to describe how to do conceptual modeling of entities and associations and contrasts it with the traditional approach to modeling. He discusses additional features of CMLs and shows how they can be used to model the dynamic aspects of an enterprise. He finishes by discussing computer aids associated with CMLs.

[BRUN88A] Glenn Bruns & Colin Potts.
Domain Modeling Approaches to Software Development.
Technical report STP-186-88, Microelectronics and Computer Technology Corporation, Austin, TX, June, 1988.
Annotation: This paper attempts to define several terms that are fundamental to

domain modeling, which is a larger activity that includes domain analysis. An overview is made of five software design approaches and their relation to domain modeling: Booch's object oriented design, JSD, Draco, Gist, and RML. Each design approach is evaluated through a series of questions for its capabilities in (1) modeling primitives, (2) domain analysis, (3) analysis/validation of the domain model, (4) specification, and (5) implementation. The same example, one of scheduling meetings among people with arbitrary schedules, is used for comparison throughout. The paper briefly discusses two additional systems which relate domain modeling to artificial intelligence knowledge acquisition, Fickas' KATE, and the MIT Requirements Apprentice. Finally, some conclusions are made about the general nature of domain abstraction mechanisms.

[CARL87A] Rick Carle.
Reusable Software Components for Missile Applications.
In *Proceedings: Tenth Minnowbrook Workshop on Software Reuse*, Syracuse University and University of Maryland, Blue Mountain Lake, NY, July, 1987.
Abstract: During 1986, Raytheon Missile Systems Division conducted an independent research project on the subject of reusable software components for missile applications. The objective was to develop and model the requirements for a software composition system based on reusable components for a software composition system based on reusable components for a limited application domain, missile systems software. A domain analysis was conducted: Missile Systems Division projects were studied and reusable software components identified. A model of a reusable software library was seeded with these components, some library access tools were built, and the concept was demonstrated.

[COHE89A] Joel Cohen.
Software Reuse for Information Management Systems.
In *Position Papers of the Reuse in Practice Workshop*, Software Engineering Institute, Pittsburgh, PA, July, 1989.
Annotation: This position paper discusses the Imagery Information Management System (IIMS) Reuse project. To reduce the cost of building this type of complex information management system the decision was made to produce a domain model for these applications, a generic architecture, a classification scheme for storing reusable components, and a populated library of such components. In doing so the project would evaluate and document several aspects of the process and investigate tools and techniques for applying reusable software. The domain analysis methodology used was that developed by Gish and Prieto-Diaz. The GTE ALS system was used to store the resulting components. The work performed was done at a high level to maximize the breadth of experience. Some conclusions about the domain analysis process are presented, and some recommendations are made for improvements in future work.

[DIET87A] Scott R. Dietzen & William L. Scherlis.
Analogy in Program Development,
The Role of Language in Problem Solving 2, Pages 95-118. North Holland, 1987.
Abstract: Consensus seems to be emerging concerning new process models for knowledge-based programming tools; surprisingly little progress has been made, however, in understanding the nature of the programming knowledge that is to be represented and organized in these tools. We approach this question by considering the role of past experience in programming, including immediate past experience, as applied to program modification, and more remote experience, as applied to new programming problems. It is necessary to represent this experience in a way that truly permits reuse of designs, without forcing direct reuse of code or even abstracted code. We

suggest that this can be achieved by considering program derivations, which represent idealized program design histories. We illustrate how abstractions on this kind of structure can support a rich space of generalizations and analogies, and we speculate on how program derivations might be represented in a semantically based programming tool.

[DIPP89B] Richard S. D'Ippolito.
Using Models in Software Engineering.
In *Proceedings of Tri-Ada '89*, Pages 256-265. Association for Computing Machinery, New York, NY, October, 1989.

Abstract: The Software Engineering Institute (SEI) has participated in several projects in which the focus was on helping contractors make use of good software engineering methods and Ada. During this participation we have learned several important lessons about the development of software for both large-scale and embedded systems. We have noticed that after a long period of time where the focus on productivity generated searches for new methodologies, tools, and ways to write reusable software, the emphasis has shifted to quality, in recognition of the fact that the new methods and tools were not adequate to address the problems occurring at the design level. We propose that the industry instead concentrate the search for the *old* methods still in use in the other branches of engineering, and apply those methods to the software problem.

[DOWL89A] S. W. Dowle.
Domain Modelling for Requirements Specification.
In *Proceedings of the Third International Conference on Command, Control, Communications and Management Information Systems*, Pages 1-7. IEE, London, UK, May, 1989.

Abstract: The Logical Systems Description (LSD) project has been established to produce requirements specifications of future submarine command systems. This paper describes one particular aspect of this strategy, namely the development and use of reference models. The paper begins by outlining the LSD project strategy and goes on to describe why reference models are necessary. The development of explicit reference models takes place during domain analysis and is dealt with in some detail, describing the method used together with its application by the LSD project to produce five domain specific reference models. The use of these models by the project is illustrated, taking as an example the elicitation activity. Finally the LSD project experience is reviewed.

[DUBO86A] Eric Dubois, et al.
A Knowledge Representation Language for Requirements Engineering.
Proceedings of the IEEE, vol. 74, no. 10: pp. 1431-1444, October, 1986.

Abstract: Requirements engineering, the phase of software development where the users' needs are investigated, is more and more shifting its concern from the target system towards its environment. A new generation of languages is needed to support the definition of application domain knowledge and the behavior of the universe around the computer. This paper assesses the applicability of classical knowledge representation techniques to this purpose. Requirements engineers insist, however, more on natural representation, whereas expert systems designers insist on efficient automatic use of the knowledge. Given this priority of expressiveness two candidates emerge: the semantic networks and the techniques based on logic. They are combined in a language called the ERAE model, which is illustrated on examples, and compared to other requirements engineering languages.

[FICK87B] Stephen Fickas.
Automating the Specification Process.
Technical report CIS-TR-87-05, University of Oregon, Eugene, Oregon, December, 1987.
Abstract: Recent research results in formalizing and automating software specification move us closer to a computer-based specification assistant. In this paper, we review three projects that we believe are particularly relevant to this goal. For each project we describe first the underlying model, and second our efforts to study it by construction of a prototype tool based on the model. Finally, we discuss incorporating the results of our study into a knowledge-based system that assists in the construction of a formal specification.

[FINK88A] Anthony Finkelstein.
Re-use of Formatted Requirements Specifications.
Software Engineering Journal, vol. 3, no. 5: pp. 186-197, September, 1988.
Annotation: The article describes techniques of searching for reusable requirements based on analogical reasoning. While the article is closely tied to a tool, it does present several techniques for formatting and isolating reusable requirements. The article provides background for representation of domain models and information for library organization.

[FISH86A] Gary Lee Fisher.
A Software Architecture for Strategic Management Support Systems Utilizing Interactive Graphics.
PhD thesis, University of Arizona, 1986.
Abstract: The author developed a software architecture for strategic-management support-systems, with underlying principle that new additions to the library of planning tools in such support systems should not have to be new programming efforts. The current status of group decision support is surveyed and the lack of a software architecture for such systems is noted. The software architecture that has been developed is intended to guide the development of such support systems and is based on a library of procedural abstraction called elemental-engines. Selected sets of elemental-engines are assembled into synthesized support drivers which support an even higher level of abstraction, that of the generic logic supporting a family of planning tools. Thus, a family of planning tools may be expanded by the simple creation of text files, containing the dialogue of the new tool. The work looks first at the nature of strategic management decision-making, then to work done in group decision support systems. A framework for software development, particularly in the area of list-processing is presented. A data structure to support such list processing is developed and discussed. An example of the software architecture is presented via the code for the initial planning-tool developed. This code was then generalized into the library of elemental-engines and a set of synthesized support drivers. This library of planning tools, built around the architecture is described, and the use of the tool in a planning session is evaluated. Some possible extensions with respect to a decision laboratory are suggested. The laboratory incorporates features developed in the evolution of using computers to support human decision-making, with software written according to the architecture presented.

[FREE87A] Peter Freeman.
A Conceptual Analysis of the Draco Approach to Constructing Software Systems.
IEEE Transactions on Software Engineering, vol. SE-13, no. 7: pp. 830-844, July, 1987.
IEEE. Reprinted with permission.
Annotation: Draco is a black box that takes system specifications written in high-

level languages (domain languages) and produces executable (or compilable) code. It can be viewed as: (1) a system constructor that uses pre-existing components, (2) a generator of program generators, or (3) a transformer of specification into executable code. Draco strives to meet external objectives (such as cost, size, etc.) through attainment of software engineering principles (i.e., abstraction, cohesion, etc.) and shows their relationship to Draco mechanisms that include multiple high-level languages, components and attributes, transformation, and refinements.

[GIDD84A] Richard V. Giddings.
Accommodating Uncertainty in Software Design.
Communications of the ACM, vol. 27, no. 5: pp. 29-35, May, 1984.

Annotation: In this article, the author claims the waterfall model of the software life-cycle does not adequately model the development process for many classes of software. He proposes a new scheme for classifying software based on the interaction between the 'universe of discourse' (the class of problems to be computed) and the actual software produced. Based on the classification scheme, he divides software into two classes: domain dependent and domain independent. Domain independent software has the following characteristics: (1) Developed under a contract with predetermined specifications; (2) No ongoing responsibility for the developer other than bug fixes; and (3) Possible limited interdependence between the software and the universe of discourse. Domain dependent software is further divided into two classes called experimental and embedded. Domain dependent experimental software is characterized by intrinsic uncertainty about the universe of discourse whereas domain dependent embedded software is characterized by an interdependence between the universe of discourse and the software. The author goes on to suggest a model for the domain dependent software life-cycle with a well-defined structure and products.

[GILR89A] Kathleen A. Gilroy, Edward R. Comer, J. Kaye Grau & Patrick J. Merlet.
Impact of Domain Analysis on Reuse Methods.
Final Report C04-087LD-0001-00, U.S. Army Communications-Electronics Command, Ft. Monmouth, NJ, November, 1989.

Abstract: The purpose of this effort is to analyze the domain analysis process and its relationship to the development and use of reusable components. It includes an identification of the critical issues and risks involved throughout the process. Capabilities of automated tools which could be used to perform various aspects of a domain analysis are also investigated. Finally, the study provides a set of guidelines for conducting a domain analysis for embedded systems. The initial work performed under this effort is a survey of existing and emerging methods and tools for performing a domain analysis and applying its results. Based on the benefits and shortcomings of existing approaches, alternative approaches to domain analysis for Army Ada applications are analyzed, and the most promising selected for further development. A consistent, cohesive, and complete methodology for domain analysis which addresses the major issues is presented. The postulated methodology for domain analysis is based on an object oriented paradigm. A three-phased approach is recommended for domain analysis: (1) Model the domain, (2) architect the domain, and (3) develop software component assets. A new concept introduced is *adaptation analysis*, i.e., the identification of differences among application systems in the domain. Automated capabilities which support the domain analysis process are proposed. These capabilities address the application of existing tools to domain analysis, as well as future tool developments. This effort identifies and addresses the key technical areas which affect the automation of domain analysis. These areas include knowledge acquisition and

knowledge-based guidance, domain languages and language-based processing, information models and data storage and retrieval, and tool and environment integration.

[GOMA90A] Hassan Gomaa.
A Domain Requirements Analysis and Specification Method.
Technical report , George Mason University, Fairfax, VA, February, 1990.
Draft.

Abstract: This paper describes a Domain Requirements Analysis and Specification Method for analyzing and specifying a family of systems. This method addresses the issues of how to represent similarities and differences in the application domain. It supports a method for generating target system specifications given the requirements of the individual target system. The goal is to provide a more effective way of managing system evolution and addressing software reuse from a generation technology perspective. The method is illustrated by means of an example.

[GOOD83A] Michael Goodell.
Quantitative Study of Functional Commonality in a Sample of Commercial Business Applications.
In *Proceedings of the Workshop on Reusability in Programming*, Pages 279-286. ITT Programming, Stratford, CT, September, 1983.

Abstract: This study offers a more precise account of the nature of functional commonality in the commercial business application domain than has been available so far. Sixty-four commercial program products were selected for study. My goals were to (1) identify recurring program functions, (2) tabulate their frequencies of occurrence, and (3) compare the function frequency distributions of products designed for different kinds of industries and applications. Frequency distributions were found to be roughly similar for all groups, with reporting always predominating, followed by either file updating or file building and copying functions.

[GREE88A] Sol J. Greenspan, Clement L. McGowan & M. Chandra Shekeran.
Toward an Object-Oriented Framework for Defining Services in Future Intelligent Networks.
In *Proceedings of the IEEE International Conference on Communications '88: Digital Technology - Spanning the Universe*, Pages 867-873. IEEE, New York, NY, June, 1988.

Abstract: The authors propose that an environment for defining services must be domain-specific (as opposed to a general-purpose programming environment). The cornerstone of such an environment is an explicit model of the network-services domain. This domain model captures the object universe of both the network capabilities and the user environment (e.g., line, call, customer), within which services are defined. A framework for object-oriented conceptual modeling that applies knowledge representation techniques from artificial intelligence is proposed as the basis. In this framework, service behavior is defined in terms of the operations associated with the domain model objects. This can accommodate basic service elements (BSEs) mandated by open network architecture (ONA) plans. Further, service definitions themselves are treated as reusable objects and organized in a taxonomic hierarchy. The goal is to define new services via extensive reuse that incorporates existing, well-tested service definitions.

[HARA89A] Mehdi T. Harandi & Mitchell D. Lubars.
Automating Software Specification and Design.
Technical report STP-001-89, Microelectronics and Computer Technology Corporation, Austin, TX, January, 1989.

Abstract: Software specification and design can be viewed as parallel activities in

which transmitted specifications are used to guide software design and the state of the design provides feedback as to the quality of the specifications. Such a paradigm can be realized with a user supplying the specifications and a knowledge-based design assistant helping to generate the software design from those specifications using a knowledge base of reusable design schemas and refinement rules. IDeA was developed as a prototypical knowledge-based design assistant to demonstrate the paradigm. This paper describes the concepts of the paradigm and the features of IDeA.

[HRYC87A] Tomas Hrycej.
A Knowledge-Based Problem-Specific Program Generator.
ACM SIGPLAN Notices, vol. 22, no. 2: pp. 53-61, February, 1987.
Abstract: A commercially relevant paradigm for a knowledge-based problem-specific program generator is described. It is applicable to repeatedly implemented software, particularly in the intermediate area between exclusive and standard software. Not the first implementation, but the repeated ones are supported using the 'problem-specific programming knowledge collected' during the past implementations. The technical details, as knowledge representation and functional component, are explained and illustrated on an example. Finally, conditions for an efficient application of the program generator are defined. The system can be developed at a relatively low expense of several months.

[HUTC88A] J. W. Hutchinson & P. G. Hindley.
A Preliminary Study of Large-Scale Software Re-use.
Software Engineering Journal, vol. 3, no. 5: pp. 208-212, September, 1988.
Annotation: The article discusses a domain analysis performed by studying existing software and determining whether components were reusable; either as is, with modification, or not at all. The methodology is that of a *commonality study*, examining existing software in light of its applicability to later developments. The study used a cataloging scheme to catalog and retrieve components. They could not attest to the success of their library effort due to the small number of components and inadequate cataloging.

[ISCO88A] Neil Iscoe.
Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach,
In Will J. Tracz, *Software Reuse: Emerging Technology*, Pages 299-308. IEEE Computer Society, Washington, DC, 1988.
Annotation: This article describes previous work and current directions at the University of Texas in software reuse within specific application domains. Previous work includes a prototype configuration system for reconfigurable databases, and a commercial screen and constraint system for microcomputer applications. Currently a model of application domain knowledge is being created for the purpose of specifying and generating programs. The overall approach is based on an object-oriented style of structuring, a visually-oriented, end-user interface, and a knowledge-based mechanism for transforming requirements into primitive functions. The article lays out a 9-step long-range research plan for addressing these issues and discusses the approach for each step in detail. The steps are: (1) Create a model of domain knowledge; (2) Implement the model; (3) Instantiate the system for the library domain; (4) Specify and generate programs within the domain; (5) Instantiate the system for another related domain; (6) Refine the model; (7) Compare the instantiations; (8) Identify the characteristics and traits that generalize across the domains, and (9) Identify the algorithms and techniques that can be used across a class of domains.

[ISCO89A] Neil Iscoe, James C. Browne & John Werth.
Generating Domain Models for Program Specification and Generation.
Technical report , University of Texas, Austin, TX, July, 1989.

Abstract: This paper describes a formalized domain modeling technique which we have designed to represent relevant portions of the domain knowledge required to specify and implement application program generators. Successful application generators are always domain specific. Software engineering of application generators and meta-generators must be based on a characterization of domain knowledge which captures the relevant application domain knowledge necessary for the paradigm. The operational goal is to allow designers, who are neither computer programmers nor domain experts, to construct application programs by declaratively describing and refining specifications for the programs that they wish to construct. The focus is on a representation technique to support the meta-level aspects of such a system- a generator for narrow domain application program generators. A domain model generated using this technique is based on classes that are composed of attributes defined in formal terms that characterize an application domain. Unique to this technique is the integration of domain-specific properties such as units, quantities, granularity, qualitative scales, population parameters into a coherent system of reusable attributes and classes. A system prototype, Ozym, has been developed to experiment with domain model generation.

[JAWO90A] Allan Jaworski, Fred Hills, Thomas A. Durek, Stuart Faulk & John E. Gaffney.
A Domain Analysis Process.
Interim Report 90001-N (Version 01.00.03), Software Productivity Consortium, Herndon, VA, January, 1990.

Abstract: This paper describes a process for domain analysis, an approach to the analysis and structuring of software requirements aimed at facilitation reuse of software requirements, design, code, and test information across a domain, a family of similar problems. Domain analysis is the critical front-end activity associated with the Software Productivity Consortium's vision of Synthesis, a process for software development that emphasizes the automated generation of software systems from software components and models designed for reuse. The major thesis of this paper is that, if we develop standard high-level designs for software systems that are not likely to change from implementation to implementation and routinely use them as frameworks for structuring requirements and lower-level design knowledge, then over time we will build up an infrastructure that supports reuse of software.

[KANG89A] Kyo C. Kang, et al.
Results from Domain Analysis Working Group.
Reuse in Practice Workshop, Pittsburgh, PA.

Annotation: The working group focused on the formulation of a general domain analysis model that could be accepted as a baseline by the software reuse community. This paper describes the Pittsburgh Workshop Model of Domain Analysis (PWMDA) which contains three parts: (1) a problem space, (2) a solution space, and (3) a mapping between the two. The problem space deals with *principles*, *features*, *relationships*, and *analogies* in the application domain. The solution space deals with the *issues*, *decisions*, and *architectural* components involved in implementing systems that match the problem requirements. Architectural components may vary greatly in level of detail from simple assertions to a detailed design description.

[KANG89B] Kyo C. Kang.
Features Analysis: An Approach to Domain Analysis.
In *Position Papers of the Reuse in Practice Workshop*, Software Engineering Institute, Pittsburgh, PA, July, 1989.
Abstract: A domain analysis was performed at the Software Engineering Institute as part of a reuse experiment. The analysis was called features analysis because of its heavy emphasis on the analysis of functional features. The goal of the analysis was to identify and represent a generalized functional model from which software requirements can be derived and based on which reusability of components can be evaluated and classification of components can be made. Some of the experiences from the analysis are: (1) the domain analysis provided opportunities for experts to consolidate and organize their domain knowledge and for non-experts to learn about the domain, (2) analyzing the functional features was an effective way to determine the product commonality and the scope of the domain analysis, and (3) there is no adequate mechanism for representing a domain model to support reuse through the requirements analysis phase. The purpose of the domain analysis was to investigate the concept and feasibility, and there was no 'formal' approach that was followed. A conceptual modelling method which is based on the analysis of the 'universe of discourse' is proposed in this paper as a domain analysis method.

[LATO89B] Larry Latour.
Issues Involved in the Content and Organization of Software Component Information Bases: Interim Report.
Technical Report for U.S. Army CECOM , University of Maine, Orono, ME, May, 1989.
Abstract: We have been tasked to investigate new and innovative techniques for organizing a database of reusable components. In this report we discuss hypertext as a tool for describing taxonomies of Ada packages in order to facilitate their reuse. Our basic premise is that the primary inhibitor to the reuse of software components is understanding. We describe a component as an information 'web' of attributes, containing specification, implementation, and usage information. The hypertext model is used to describe component information webs, alternate taxonometric structures leading to these webs, and class information webs describing information common across component webs. To better understand the content of information webs we discuss a number of related topics such as the relationship between design and reuse, the notions of context independent and context dependent information, and the relationship of these notions to domain analysis. Included is a description of a reuse experiment performed at Maine using the Booch components.

[LEDB83A] Lamar Ledbetter.
Reusability of Domain Knowledge in the Automatic Programming System (phi).
In *Proceedings of the Workshop on Reusability in Programming*, Pages 97-105. ITT Programming, Stratford, CT, September, 1983.
Abstract: There are two primary issues in software reusability. The first is the ease of incorporating an existing software entity in new applications. In this issue, the concerns are primarily with algorithm performance, linkage, parameter typing and order, output values and side effects. The second issue is the ability to define a computational problem or sub-problem and retrieve the software entities which either solve the problem or can be easily modified to solve the problem. This paper discusses the potential impact on these software reusability issues of automatic programming systems which operate on computational models defined in user domain terms.

[LEE88A] Kenneth J. Lee, et al.
An OOD Paradigm for Flight Simulators, 2nd Edition.
Technical report CMU/SEI-88-TR-30, Software Engineering Institute, Pittsburgh, PA, September, 1988.
Annotation: This report presents a paradigm for object-oriented implementations of flight simulators. A jet engine simulator was implemented. The approach used to create the simulator was to represent the various engine components as objects. These objects then communicated in ways analogous to those used among the different components of an actual jet engine. This allowed the overall behavior of the engine to be accurately simulated based on the theoretical engineering models for each of the objects (i.e., burner, rotor, diffuser, etc.). The resultant paradigm produced is in some sense a domain analysis for jet engines. Reusable code templates were used to standardize the object interfaces, and contained the general features and placeholders for the specific features of the objects. This report represents work done on the Ada Simulator Validation Program (ASVP) carried out by members of the technical staff at the Software Engineering Institute (SEI).

[LUBA87A] Mitchell D. Lubars.
A Knowledge-based Design Aid for the Construction of Software Systems.
PhD thesis, University of Illinois at Urbana-Champaign, 1987.
Abstract: This thesis addresses the use of knowledge-based techniques in providing high-level support for software design activities. A knowledge-based refinement paradigm of software development is introduced that alleviates some of the problems of the traditional software life-cycle. A design system, IDeA, is presented as a prototypical environment that supports this paradigm. Within IDeA, design knowledge is encoded in schematic forms that abstract out the common design features across related application domains. These *design schemas* represent design families and provide significant potential for design reuse, since they can be customized and refined to satisfy a particular user's design requirements. This is accomplished through the application of specialization and refinement rules that are associated with the schemas. Design schemas also represent shared constraints and design decisions. This schema representation is based on a notion of polymorphic data typing with multiple constraints. IDeA supplements the knowledge-based techniques with a schema selection strategy that facilitates user selection of design fragments. In addition, constraint propagation and planning techniques provide support for intelligent design activities. These techniques are integrated with other support aspects, such as clerical, organizational, and analysis support, to present IDeA as a complete environment for software design.

[LUBA88A] Mitchell D. Lubars.
A Domain Modeling Representation.
Technical report STP-366-88, Microelectronics and Computer Technology Corporation, Austin, TX, November, 1988.
Abstract: A serious problem in most software development projects is that key information fails to get recorded, and many other kinds of information are not uniformly integrated. A possible solution to the problem is to evolve a model of the software system that includes the relevant knowledge about its development. It is conjectured that if all this information could be recorded, its use would provide better quality software, facilitate communication between software engineers, enhance software maintenance, and provide software artifacts that are more reusable. This paper presents a domain modelling representation (DMR) that represents a large portion of the desired information.

[LUBA88B] Mitchell D. Lubars.
Domain Analysis and Domain Engineering in IDeA.
Technical report STP-295-88, Microelectronics and Computer Technology Corporation, Austin, TX, September, 1988.
Annotation: This paper describes an approach to domain analysis based on the information contained in the IDeA knowledge bases. The information is classified into six categories. The first category is *properties* about objects (objects can be anything) in the domain. These properties are mainly attributes describing the objects. The properties are arranged in a tree structure which has the more abstract objects appearing higher in the tree. The relations between the properties are derived from the context of the domain. *Data Types* are described in terms of their properties and are organized into a type lattice. The data type lattice is also used in classifying and selecting design schemas. *Design Schemas* are abstract solutions for a class of related design problems. Although not stated, these appear to be organized into an abstraction hierarchy. *Schema specialization rules* are mappings between an abstract design schema and a more specialized design schema. Schema specializations are almost always associated with the addition of requirements or design commitments beyond those implied by the more abstract design schema. *Schema refinement rules* are mappings between a design schema and a data flow design that represents the refinement or implementation of that schema. Refinement rules don't change the level of abstraction. *Type constraints* are included in design schemas to propagate property assignments of data types to other data types that share the same abstract property class. The domain analysis process is defined as a set of seven steps and the domain engineering process is defined as a set of nine steps.

[MANO88A] Nobuoki Mano.
Modeling of Data-Processing Software for Generating and Reusing Their Programs.
In *Proceedings of the 10th International Conference on Software Engineering*, Pages 231-240. IEEE, New York, NY, April, 1988.
Annotation: A modeling scheme named S-model (semantic model) is proposed, which is based on a uniform object-relationship formalism. S-model combines notions from logic, set theory, and abstract syntax and covers a wide range of information on file-processing and data-structure-manipulating software. With the procedural algorithms and problem-solving capability of the S-model system, it is possible to generate and reuse programs in various fields.

[MAUR88A] John Mauri, Celeste N. Anton & Mark L. McLeod.
Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design.
Final Technical Report RADC-TR-87-118, Rome Air Development Center, Griffiss AFB, NY, January, 1988.
Abstract: A functional description specification and functional prototype were developed. The functional description recommends a design approach wherein functions result from a decomposition of cartographic processes and algorithms into generic, building block, low-level primitives. These low-level routines are then aggregated to create higher-level cartographic functions and applications which can in turn be combined to form systems. The functional software prototype demonstrated the feasibility and advantages of this design approach and will be used as a baseline upon which to evolve the final design for a generic cartographic application.

[MCCA85C] Ron McCain.
A Software Development Methodology for Reusable Component.
In *Proceedings of the Software Technology for Adaptable Reliable Systems (STARS) Workshop*, Pages 361-384. Naval Research Laboratory, Washington, DC, April, 1985.
Available from Defense Technical Information Center as AD-A163 463.
Abstract: Software reusability could potentially provide substantial economic benefits. Large-scale software component reuse, however, will not be possible without a software development approach that emphasizes the production of reusable software components. This paper defines the characteristics of reusable software and proposes a software development methodology that produces software components exhibiting these characteristics. The methodology is intended to supplement rather than replace other sound software development methodologies. In addition to describing the reusability-oriented thought process associated with the methodology, the paper suggests new work products and validation procedures to support the methodology.

[MCNI86A] Daniel G. McNicholl, et al.
Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results.
Technical report AFATL-TR-85-93, McDonnell Douglas Astronautics Company, St. Louis, MO, May, 1986.
Abstract: The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts cataloguing scheme and parts composition system were developed to support parts usage.

[MCNI88A] Daniel G. McNicholl, Sholom G. Cohen, Constance Palmer, et al.
Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System.
Final Report AFAL-TR-88-62, Vol. I, McDonnell Douglas Astronautics Company, St. Louis, MO, November, 1988.
Abstract: CAMP-2 was primarily a technology demonstration of the concepts developed in CAMP-1. The first major task was the construction of the reusable parts identified during CAMP-1. A total of 454 production-quality, reusable Ada parts were coded, tested, and documented in accordance with DoD-STD-2167. In addition, a prototype of the parts composition system (PCS) tool defined in CAMP-1 was also constructed, tested, and documented in accordance with DoD-STD-2167. To illustrate the utility of this tool, a user can spend 3 minutes describing his requirements for a Kalman filter subsystem and the tool will generate and assemble over 1900 lines of Ada code which efficiently implement this subsystem.

[MOOR88A] John M. Moore & Sidney C. Bailin.
Position Paper on Domain Analysis.
In *Position paper for Domain Analysis Working Group at OOPSLA '88*, Association for Computing Machinery, New York, NY, October, 1988.
Annotation: This position paper discusses the general domain analysis process as used by CTA in performing a Control Center domain analysis for NASA Goddard Space Flight Center. The areas covered include a high-level overview of the methodology, a pragmatic discussion of domain boundary scoping, and the way CTA represented the domain model which resulted from the analysis. The domain model consisted of (1) an interactive model (a

reuse database), and (2) a graphical model (using annotated entity-relationship diagrams). A brief discussion is also given on the process of identifying objects in the course of the domain analysis.

[MOOR89A] John M. Moore & Sidney C. Bailin.
Domain Analysis: Framework for Reuse.
Technical report , Computer Technology Associates, Rockville, MD, October, 1989.
Annotation: This paper presents a life-cycle approach to domain analysis and reuse-based software development. Domain analysis is seen as complementary and parallel to the ongoing process of system development. Reuse-based development is described in terms of both the *demand side* (seen by the user of reusable resources) and the *supply side* (seen by the developer of reusable resources, which includes domain analysis and reusable product development). Both aspects are examined in detail. The paper concludes with a discussion of the domain analysis method used for the control center domain in work done for NASA Goddard Space Flight Center.

[MOST85A] John Mostow.
Toward Better Models of the Design Process.
The AI Magazine, pp. 44-57, Spring, 1985.
Annotation: One of the key research problems in AI-based design for the near future may be to develop a better model of the design process by incorporating knowledge-based techniques. A comprehensive model of design should address the following aspects of the design process: design decisions; rationales for design decisions; control of the design process; and the role of learning in design. This article presents some of the most important ideas emerging from current AI research on design, especially ideas for better models of design. It is organized into sections dealing with each of the aspects of design listed above. The author recommends development of better models of the design process. He advocates compliance with six criteria: (1) make the state of the design explicit, (2) make the goal structure explicit, (3) make the design decisions explicit, (4) make the design rationale explicit, (5) understand how to control the design process, and (6) integrate the role of learning in design.

[MYER88A] Brad A. Myers.
A Taxonomy of Window Manager User Interfaces.
IEEE Transactions on Computer Graphics & Applications, vol. 8, no. 5: pp. 65-84, September, 1988.
IEEE. Reprinted with permission.
Abstract: This article presents a taxonomy for the user-visible parts of window managers. It is interesting that there are actually very few significant differences, and the differences can be classified in a taxonomy with fairly limited branching. This taxonomy should be useful in evaluating the similarities and differences of various window managers, and it will also serve as a guide for the issues that need to be addressed by designers of future window manager user interfaces. The advantages and disadvantages of the various options are also presented. Since many modern window managers allow the user interface to be customized to a large degree, it is important to study the choices available.

[NEIG80A] James M. Neighbors.
Software Construction Using Components.
PhD thesis, University of California at Irvine, 1980.
Annotation: This dissertation asserts that the reuse of software results only from the reuse of analysis, design, and code, rather than simply the reuse of code. The concept of domain analysis is introduced to describe the activity

of identifying the objects and operations of a class of similar systems in a particular problem domain. Experiments using a prototype system, Draco, are presented and the results discussed.

[NEIG83A] James M. Neighbors.
The Draco Approach to Constructing Software from Reusable Components.
In *Proceedings of the Workshop on Reusability in Programming*, Pages 167-178. ITT Programming, Stratford, CT, September, 1983.
Abstract: This paper discusses a mechanism called Draco which aids in the construction of software systems. In particular we are concerned with the reuse of analysis and design information in addition to programming language code. The goal of the work on Draco has been to increase the productivity of software specialists in the construction of *similar* systems. The particular approach we have taken is to investigate the construction of software from reusable software components which are organized by problem domain. The experimental method used was to hypothesize a scheme based on previous work and experiment with example problems on a prototype system.

[NEIG87A] James M. Neighbors.
Report on the Domain Analysis Working Group Session.
In *Proceedings of the Workshop on Software Reuse*, Rocky Mountain Institute of Software Engineering, Boulder, CO, October, 1987.
Abstract: This working group report touches briefly on a number of issues central to domain analysis. It discusses some rationales for performing domain analysis, various representation techniques generally proposed for domain analysis, some of the results of a simple test application of domain analysis to a small example system specification, and a brief outline of the domain analysis process used.

[PARN76A] David L. Parnas.
On the Design and Development of Program Families.
IEEE Transactions on Software Engineering, vol. SE-2, no. 1: pp. 1-9, March, 1976.
IEEE. Reprinted with permission.
Abstract: Program families are defined (analogously to hardware families) as sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members. The assumption that, if one is to develop a set of similar programs over a period of time, one should consider the set as a whole while developing the first three approaches to the development, is discussed. A conventional approach called 'sequential development' is compared to 'stepwise refinement' and 'specification of information hiding modules'. A more detailed comparison of the two methods is then made. By means of several examples it is demonstrated that the two methods are based on the same concepts but bring complementary advantages.

[PARN79A] David L. Parnas.
Designing Software for Ease of Extension and Contraction.
IEEE Transactions on Software Engineering, vol. SE-5, no. 2: pp. 128-138, March, 1979.
IEEE. Reprinted with permission.
Abstract: Designing software to be extensible and easily contracted is discussed as a special case of design for change. A number of ways that extension and contraction problems manifest themselves in current software are explained. Four steps in the design of software that is more flexible are then discussed. The most critical step is the design of a software structure called the 'uses' relation. Some criteria for design decisions are given and illustrated using a small example. It is shown that the identification of

minimal subsets and *minimal* extensions can lead to software that can be tailored to the needs of a broad variety of users.

[PARN85A] David L. Parnas, Paul C. Clements & David M. Weiss.
The Modular Structure of Complex Systems.
IEEE Transactions on Software Engineering, vol. SE-11, no. 3: pp. 259-266, March, 1985.
IEEE. Reprinted with permission.

Annotation: The authors noticed a growing gap between software engineering principles being advocated at major conferences and the practice of software engineering at many industrial and governmental laboratories. They saw that either good ideas were illustrated using unrealistically simple code fragments or complex problems were not worked out in any great detail. They decided to take a realistic, difficult problem, apply *academic* ideas, and improve it. Success would show: (1) the feasibility of the ideas, (2) a model for other developers, and (3) how to refine ideas to work in more complex situations than those described in the literature. The problem selected was the operational flight program (OFP) for the A-7E aircraft. This program used many 'dirty tricks', barely fitted in memory, and barely met its real-time constraints. It was also considered to be one of the best programs of its type, and could be considered sufficiently challenging so that skeptics would not attribute success to poor quality of the original program. The original program could be viewed as one big module. The team decided to decompose the program into separate modules based on the idea of information hiding. This would make the program more maintainable by allowing modules to be designed and revised independently. The teams experience suggested that the use of information hiding in complex systems is practical. The team created what they called a 'module guide' which is a sort of description of the OFP domain. This document proved to be useful to designers and programmers when attempting to resolve problems. It also aided new programmers joining the project in understanding the structure of the program.

[PAYT88B] Teri F. Payton.
Reusability Library Framework.
Presentation at STARS Foundations Workshop , Unisys Defense Systems, Paoli, PA, April, 1988.
Reprinted with permission of Unisys Corporation.

Annotation: The Unisys Reusability Library Framework (RLF) STARS Foundations project is intended to provide a general framework and a base set of tools supporting the creation and maintenance of a repository of reusable Ada software components, organized around particular application domains. This paper presents a brief summary of the RLF project. Unisys believes that the most effective gains in productivity through reuse will be from the development of libraries of components for specific domains, which are structured according to explicit *domain models*. The objective of the RLF project is to develop knowledge-based foundations technology for building *intelligent librarians,* and to demonstrate the use of this technology by building an example library for the domain of Ada benchmark tests. The underlying knowledge representation used is based on semantic networks and is implemented in the AdaKNET subsystem.

[PERR89A] James M. Perry & Mary Shaw.
The Role of Domain Independence in Promoting Software Reuse: Architectural Analysis of Systems.
In *Position Papers of the Reuse in Practice Workshop*, Software Engineering Institute, Pittsburgh, PA, July, 1989.

Abstract: While there are several variations of domain analysis, they are usually

characterized by their emphasis on application dependencies. This position paper describes *architectural analysis* which is a type of analysis for furthering our understanding of software architectures. It attempts to raise the abstraction level of design elements and, thereby, emphasizes domain independence. Although architectural analysis and domain analysis for reuse have different processes and goals, they are closely related and support one another. This mutual support is identified and examined. The SEI Software Architecture Project is described to provide an example of architectural analysis.

[POLL85A] Guylaine Marie Pollock.
A Design Methodology and Support Environment for Complexity Metrics via Reusable Software Parts.
PhD thesis, Texas A&M University, 1985.
Abstract: The objective of this research is the design of a system of reusable software parts which can be utilized as kernel primitives in constructing prototype software metrics. A methodology for designing an automatable set of elementary measures of primitive constructs is first defined. Primitives designed through this methodology are used to develop a library of reusable software parts for implementation of composite or hybrid software metric prototypes. A domain analysis of software complexity metrics served to define basic operations or functional primitives common to complexity metric definitions. Twenty-two representative software complexity metrics were analyzed to determine common measurements or functions between the various metrics. In addition, basic data structures such as stacks, queues, and linked lists that were needed by the kernel primitives were also identified. A kernel set of metric primitives was defined from this analysis. Generic Ada packages were designed to implement the defined structures and the operations necessary for their manipulation. The delineated functions compose a reusable software library. The system provides an attractive research environment for software metric studies. Reusability aspects of the design and partial implementation support development of rapid prototypes. The Ada language was useful in providing constructs to facilitate implementation of the reusable software parts. Separate compilation, generics and tasking were attractive features. Strong typing restrictions, however, limited the design capabilities. Based on experiences in this research the methodology appears to be reusable in developing metric primitives for other metric classes.

[PRIE87C] Ruben Prieto-Diaz.
Domain Analysis for Reusability.
In *Proceedings of COMPSAC 87:The Eleventh Annual International Computer Software & Applications Conference*, Pages 23-29. IEEE Computer Society, Washington, DC, October, 1987.
Annotation: This paper proposes a methodology for domain analysis. The process is decomposed into three areas: pre-DA activities, DA, and post-DA activities. Pre-DA includes: defining and scoping the domain, identifying sources of knowledge and information about the domain, and defining the approach to DA. Post-DA activities include: identification and implementation of reusable components, and production of software reuse guidelines. The paper defines the *context* for DA as the inputs and output to DA. The inputs are: DA guidelines from the domain analyst, domain knowledge from the domain expert, and standard examples from existing systems. The outputs from DA are: reusable components and domain standards. The process is given in data flow diagrams where the three main steps are: prepare domain information, analyze domain, and produce reusable work-products.

[PRIE87D] Ruben Prieto-Diaz.
Faceted Classification and Reuse Across Domains.
In *Proceedings of the Workshop on Software Reuse*, Rocky Mountain Institute of Software Engineering, Boulder, CO, October, 1987.
Annotation: Most reusability successes result from concentrating on a narrow domain, but little has been done about reuse across domains. This paper proposes an approach to facilitate reuse across domains using concepts from domain analysis and faceted classification. This approach uses domain analysis to derive faceted classification schemes of domain specific collections (via *literary warrant* (LW)), and then derives a global faceted scheme that relates the different domain specific vocabularies. A global scheme allows users to identify and select components from different application domains and thus increase their potential reusability.

[PRIE89A] Ruben Prieto-Diaz.
Domain Analysis Tutorial.
In *Papers from the Workshop: Tools and Environments for Reuse*, Software Productivity Solutions in cooperation with the IEEE Committee on Software Engineering, Indialantic, FL, May, 1989.
Annotation: This tutorial presentation presents an overview of many of the fundamental aspects of domain analysis, from definitions of domain analysis terminology to general background including the Draco domain analysis process, CAMP, Prieto-Diaz's domain analysis model, Arango's work formalizing domain analysis, the University of Oregon KATE system work, and the CTA work for NASA Goddard.

[PUNC88A] P. Paolo Puncello, Piero Torrigiani, Francesco Pietri, Riccardo Burlon, Bruno Cardile & Mirella Conti.
ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System).
IEEE Software, vol. 5, no. 2: pp. 58-66, March, 1988.
Abstract: This article reports on ESPRIT Project 401, building the Application Software Prototype Implementation System. ASPIS exploits artificial intelligence techniques in a software-development environment. Our goal is to encourage a more flexible and effective software-development life cycle, smoothing the transition between user needs, analysis, and design. The novel aspects of our project are the knowledge-based tools called assistants and the definition of a logic-based formalism for specifications. ASPIS has four assistants. Two knowledge-based assistants - an Analysis Assistant and a Design Assistant - are used directly by the developers of a particular methodology. They embody knowledge about both the method and the application domain. Once defined, the specifications can be executed by the Prototype Assistant, which verifies the system's properties. A fourth assistant, the Reuse Assistant, helps developers reuse specifications and designs.

[ROBI89B] William N. Robinson.
Integrating Multiple Specifications Using Domain Goals.
Technical report CIS-TR-89-03, University of Oregon, Eugene, Oregon, February, 1989.
Abstract: Design is a process which inherently involves tradeoffs. We are currently pursuing a model of specificaton design which advocates the integration of multiple perspectives of a system. We have mapped the intrgration problem onto the negotiation problem of many issues between many agents in order to apply known resolution techniques. Part of that mapping requires the modeling of domain goals which serve as issues for negotiation. Herein, we describe the use of domain goals in our conflict resolution process which is applied during the intrgration of specifications.

[ROSA88A] S. Roody Rosales & Prem K. Mehrotra.
MES: An Expert System for Reusing Models of Transmission Equipment.
In *Proceedings of the Fourth Annual Conference on Artificial Intelligence Applications*, Pages 109-113. IEEE, New York, NY, March, 1988.

Abstract: The *Modelling Expert System (MES)* aids in the generation of models for new transmission equipment (unit) by reusing the models of existing transmission equipment. The concept of model reuse is based on 'clones'. An existing model is said to be a base model for generating a 'clone' (the model of a new unit), if the transmission characteristics of the two models are similar but not identical. There may be some changes ('fixes') required in the base model to generate the model for the new unit. The knowledge required for deciding what existing models can serve as the basis for cloning is empirical, ill-structured, and known only to a few human experts. The expert system programming facilitates preservation of such knowledge in software. The 'clone' determination/generation knowledge for a few equipment families has been captured in MES. The models of the transmission equipment are presently written in PL/I programming language. The code changes ('deltas') that are required in the PL/I code of the base model, to generate the code for the new unit, do not require extensive knowledge of the PL/I language or modelling. Instead, these 'deltas' are specified as templates. MES also aids in the generation of these 'deltas' by using the powerful macro facility of LISP. MES is written in OPS5 and Franz LISP, and runs under AT&T UNIX operating system. The clone determination part is largely OPS5 rule-based, whereas the code generation software consists of LISP functions. MES also uses a pattern-action language, AWK, for finding the lines where the code changes are to be made.

[SCHE86A] William L. Scherlis.
Abstract Data Types, Specialization, and Program Reuse.
In *Lecture Notes in Computer Science (#244): Proceedings of an International Workshop on Advanced Programming Environments*, Pages 433-453. Springer-Verlag, June, 1986.

Abstract: It is often asserted that our ability to reuse programs is limited primarily by the power of programming language abstraction mechanisms. We argue that, on the basis of performance considerations, this is just not the case in practice- these 'generalization' mechanisms must be complemented by techniques to adapt the generalized structures to specific applications. Based on this argument, we consider a view of programming experience as a network of programs that are generalizations and specializations on one another and that are interconnected by appropriate program derivation fragments. We support this view with a number of examples. These examples illustrate the important role of abstract data type boundaries in program derivation.

[SHLA89A] Sally Shlaer & Stephen J. Mellor.
An Object-Oriented Approach to Domain Analysis.
ACM SIGSOFT Software Engineering Notes, vol. 14, no. 5: pp. 66-77, July, 1989.

Annotation: This paper presents an object-based approach to domain analysis called *object-oriented analysis*. The approach is based on building three types of formal models: an information model, a set of state models, and a set of process models. All three models are used together with prescribed rules of integration. Data flow diagrams are derived mechanically from the state models. The interactions between the state models are depicted in a graphical representation called an Object Communication Model.

[SHLA90A] Sally Shlaer & Stephen J. Mellor.
Recursive Design.
Computer Language, vol. 7, no. 3: pp. 53-65, March, 1990.
Abstract: Modern approaches to systems analysis focus largely on the particular application problem, as expressed by the system's eventual users. The analysis is expressed in terms of real-world entities such as trains, passenger stations, switches, and cargo shipments. However, to develop a software system design, the designer must also contend with computer science entities such as messages, mailboxes, tasks, and files - the conceptual entities of the implementation language and operating system. To bridge the distance between the application and the implementation, experienced designers often speak loosely of 'making a mapping' between these separate and distant worlds. Recursive design is a method for handling the transistion between analysis and design. In recursive design, the emphasis is on producing mappings that can be applied to all the elements of the problem in a uniform and systematic manner.

[SIDO89A] James L. Sidoran.
Conceptual Modeling and C3I System Design.
In *Proceedings of the AIAA Computers in Aerospace VII Conference*, Pages 672-677. The American Institute of Aeronautics and Astronautics, Washington, DC, October, 1989.
Abstract: This paper describes current research in an expert system based approach to domain analysis. Currently, expert systems are engineered with a narrow focus on the domain. This results from the fact that in order for expert level information to exist in an automated tool, it is necessary to go deep and narrow. Consequently, this restricts analysis activities. An extension to this approach is to provide a 'meta-modelling' paradigm that has initially, high-level domain specific information and terminology. In theory, as the breadth of the domain is defined, and frameworks for domain structures are incorporated into the representation, it may be reasonable to approach a domain definition that is both broad and detailed. This paper only serves to identify and bound this capability within the domain of Command and Control (C2). A prototype tool, Conceptual Modelling via Logic Programming (CMLP), has been developed containing an initial system structure shell of the Command and Control domain. The prototype tool includes a user interface that allows the user to view multiple perspectives of the domain in multiple windows.

[SIMO87B] Mark A. Simos.
The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability.
In *Proceedings of the Workshop on Software Reuse*, Rocky Mountain Institute of Software Engineering. Boulder, CO, October, 1987.
Annotation: One fundamental problem in software reuse is integrating reusability into a conventional top-down 'waterfall' life-cycle model of software development. This paper outlines some general characteristics that an extended life-cycle model should have. These characteristics include: (1) A perspective centered on the notion of 'domains', or 'families' of related programs or systems supporting particular application areas; (2) concentration on application specificity, or 'narrow-band' reuse within specific application domains as the best means of achieving significant productivity increases; and, (3) recognition that a spectrum of techniques for reusable software are needed, such as 'ad hoc' reuse, libraries, code generation techniques, and even knowledge-based techniques. The paper discusses the Application-Specific Languages (ASL) approach used at Unisys in such areas as computer system configuration and message format

translation/validation. It also discusses the need for a perspective on software development that looks beyond the single project life-cycle and allows the traceability of information across individual project phases.

[SOLD89A] James J. Solderitsch, Kurt C. Wallnau & John A. Thalhamer.
Constructing Domain-Specific Ada Reuse Libraries.
In *Proceedings of the Seventh Annual Conference on Ada Technology*, Pages 419-433. U.S. Army Communications-Electronics Command, Ft. Monmouth, NJ, March, 1989.

Abstract: High-impact reuse is achieved by focusing on specific Application Domains. A Software Component Reuse Library System must support domain modelling as well as repository management features. The RLF project addresses both of these areas. Repository management capabilities including retrieval, classification, insertion, and qualification of components are all provided. Domain modelling is achieved through knowledge representation components that were developed in Ada using an Ada perspective. The domain model provides an effective and powerful interface to the library. An evolutionary approach has enabled the production of a family of library applications of varying functionality and point-of-view. Ada features, such as generics and exception handling, and Ada design principles, such as data abstraction, are used to construct systems that incorporate traditional AI functionality while providing enhanced system maintainability and evolvability.

[UNIS88A] Unisys.
Reusability Library Framework AdaKNET/AdaTAU Design Report.
Design Report , Unisys Corporation, Paoli, PA, May, 1988.
Reprinted with permission of Unisys Corporation.

Annotation: The Reusability Library Framework (RLF) was a STARS foundation area development. RLF provides a basic set of tools to create and maintain libraries of reusable components in specific domains. The system supports knowledge-based representation of domains to assist in the selection of components. This technical report is an overview of the capabilities of the system and is useful for developing both domain and library representation techniques, but does not provide direct insight into the domain analysis process.

[VITA90B] William Vitaletti & Ernesto Guerrieri.
Domain Analysis within the ISEC Rapid Center.
In *Proceedings of the Eighth Annual National Conference on Ada Technology*, Pages 460-470. U.S. Army Communications-Electronics Command, Ft. Monmouth, NJ, March, 1990.

Abstract: One of the main activities of the RAPID Center in supporting a development effort in the identification of reuse possibilities within the system being developed as well as across similar systems. This activity is accomplished by performing a domain analysis. This paper describes the domain analysis process proposed for the RAPID Center, the expected domain model, and the experience gained from applting this process to the currently supported development effort.

[WARD88A] Paul T. Ward & Lloyd G. Williams.
Using the Structured Techniques to Support Software Reuse.
In *Proceedings of the Structured Development Forum X*, Pages 211-222. Structured Development Forum, San Francisco, CA, August, 1988.

Abstract: The development and use of reusable software components offers a means of reducing the cost of new software systems while, at the same time, improving their quality. Recent advances in software engineering technology have increased the practicality of software reuse. There is,

however, very little available in the way of support for the identification, design, and construction of reusable components. Object oriented development and domain analysis are two techniques which offer support for developing reusable components. Object oriented development assists in structuring software systems so that their components have a high potential for reuse. Domain analysis assists in identifying those components. This paper presents an approach to the development of reusable software components based on the use of Real-Time Structured Analysis and Design in conjunction with domain analysis and object oriented development.

[WEBS88A] Dallas E. Webster.
Mapping the Design Information Representation Terrain.
Technical report STP-367-88, Microelectronics and Computer Technology Corporation, Austin, TX, November, 1988.
Abstract: This report attempts to establish a context for design information representation research, via a qualitative survey and comparison of a broad range of relevant technologies. We discuss associated representation requirements, and present a working 'technology map', which reflects relationships among the surveyed technologies. Within the context provided by the map we point out limitations of the representation mechanisms of conventional software development technologies, and prospects for overcoming them. This report results from editing STP-093-087 for publication in IEEE Computer. It reflects significant reorganization and condensation, as well as some updating of the earlier, more detailed report. It also provides additional background material to make it accessible to a wider audience.

[WEIS88A] David M. Weiss.
Reuse and Prototyping: A Methodology.
Technical report SPC-TR-88-022, Software Productivity Consortium, Reston, VA, March, 1988.
Abstract: This report describes the concepts underlying a proposed methodology for software development and maintenance that encompasses both prototyping and reuse. It shows how the concepts may be embodied in tools that support the methodology, and suggests how the Consortium's current set of tools can evolve into a set that fully supports the methodology. Example scenarios of the application of the concepts and the tools are also presented. The underlying concepts discussed are information hiding, program families, hierarchical structuring by both information hiding relation and by the uses relation, and characterization of modules as black boxes. The envisioned development paradigm consists of maintaining collections of program families and tools for searching through such collections, adapting family components to create new family members, composing new family members from existing components, describing families, assessing families, and storing families, including the information needed to characterize them for future use.

[YADA88A] Surya B. Yadav, et al.
Comparison of Analysis Techniques for Information Requirement Determination.
Communications of the ACM, vol. 31, no. 9: pp. 1090-1097, September, 1988.
Annotation: The article presents a framework for analysis and comparison of requirements analysis techniques. It also discusses an experiment to apply the framework in comparing two such techniques. The article presents a good discussion of parameters for domain analysis: (1) What requirements should be; (2) how they should be represented, and (3) how they should be derived.

[YADA89A] Surya B. Yadav & D. R. Chand.
An expert modeling support system for modeling an object system to specify its information requirements.
Decision Support Systems, vol. 5, no. 1: pp. 29-45, March, 1989.
Abstract: The paper presents an analyst support system that partially automates the process of determining information requirements. The focus is upon the problems associated in developing a clear and consistent understanding of the object system by the analysis team, where the object system is essentially the larger system that an information system serves. The paper describes the design of a computer aided tool which is based upon an extension of SADT and it incorporates the use of problem domain knowledge-base and decision concepts of Simon. The emphasis is on the rationale for the structure and components of this tool and how it may be used for modeling the object system and performing both component and precedence analysis. The problem of generating information system requirements is reduced to traversing the conceptual model.

3. Classification of References

The following table lists all of the references and marks the general area of domain analysis addressed by the reference. This simple classification is intended to help someone seeking information about a specific aspect of domain analysis. There are six general categories identified here:

1. *Information Gathering*
   The process of collecting proper and sufficient information/knowledge with which to perform a domain analysis
2. *Domain Analysis (DA) Methodology*
   The methodology used to perform a domain analysis
3. *Tools and Environment Support*
   The automated tools and environment support used in performing and applying domain analysis
4. *Representation- Domain Models and Software Architectures*
   The forms in which the domain analysis information is represented and available to the user
5. *Application Domains*
   The specific "real-world" applications to which domain analysis has been applied
6. *Management Issues*
   Aspects of domain analysis which are not directly related to the technical problems, such as economic, legal, and managerial

Name	Information Gathering	DA Method	Tools & Environ.	Model/ Architect.	Application Domains	Mgmt. Issues
ADEL85A	•					
ALEX86A	•	•				
ALLE87A	•		•			
ARAN88A		•				
ARAN88B			•			
ARAN88C		•				
ARAN89A		•				
ASDJ88A			•			
BAIL88B					•	
BAIL89C					•	
BAIL89D	•		•	•		
BARS85A					•	
BATO88A				•	•	
BATO88B				•	•	
BATO88C			•		•	
BENN84A	•			•		
BIGG88B	•			•		
BORG84A				•		
BORG85A				•		

Name	Information Gathering	DA Method	Tools & Environ.	Model/ Architect.	Application Domains	Mgmt. Issues
BRUN88A				•		
CARL87A			•		•	
COHE89A					•	
DIET87A				•		
DIPP89B				•		
DOWL89A					•	
DUBO86A				•		
FICK87B	•	•		•		
FINK88A			•	•		
FISH86A				•	•	
FREE87A			•	•		
GIDD84A		•				
GILR89A	•	•	•	•		•
GOMA90A		•				
GOOD83A					•	
GREE88A				•	•	
HARA89A			•	•		
HRYC87A	•		•			
HUTC88A	•	•				
ISCO88A		•		•		
ISCO89A		•	•	•		
JAWO90A		•		•	•	
KANG89A		•		•		
KANG89B		•				
LATO89B			•			
LEDB83A					•	
LEE88A					•	
LUBA87A			•	•		
LUBA88A				•		
LUBA88B		•		•		
MANO88A				•		
MAUR88A				•	•	
MCNI86A			•	•		
MCNI88A			•			

Name	Information Gathering	DA Method	Tools & Environ.	Model/ Architect.	Application Domains	Mgmt. Issues
MOOR88A		•		•		
MOOR89A		•				
MOST85A				•		
MYER88A				•	•	
NEIG80A			•	•		
NEIG83A			•	•		
NEIG87A		•				
PARN76A		•				
PARN79A	•	•				
PARN85A					•	
PAYT88B			•		•	
PERR89A				•		
POLL85A		•			•	
PRIE87C		•				
PRIE87D			•			
PRIE89A		•				
PUNC88A			•			
ROBI89B		•		•		
ROSA88A			•		•	
SCHE86A				•		
SHLA89A		•				
SHLA90A		•				
SIDO89A			•	•		
SIMO87A			•			
SIMO87B						•
SOLD89A			•	•		
UNIS88A			•	•		
VITA90B		•		•	•	
WARD88A		•				
WEBS88A				•		
WEIS88A		•				
YADA88A				•		
YADA89A	•		•			

Appendix I
Alphabetical by Author's Name

ADELSON, BETH

- ADEL85A: The Role of Domain Experience in Software Design

ALEXANDER, JAMES H.

- ALEX86A: Knowledge Level Engineering: Ontological Analysis

ALLEN, BRADLEY P.

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

ANTON, CELESTE N.

- MAUR88A: Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design

ARANGO, GUILLERMO F.

- ARAN88A: Domain Engineering for Software Reuse
- ARAN88B: Evaluation of a Reuse-Based Software Construction Technology
- ARAN88C: Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse
- ARAN89A: Domain Analysis - From Art Form to Engineering Discipline

ASDJODI, MARYAM

- ASDJ88A: Knowledge-Based Component Composition: An Approach to Software Reusability

BAILIN, SIDNEY C.

- BAIL88B: Semi-Automatic Development of Payload Operations Control Center Software
- BAIL89C: Generic POCC Architectures
- BAIL89D: The KAPTUR Environment: An Operations Concept
- MOOR88A: Position Paper on Domain Analysis
- MOOR89A: Domain Analysis: Framework for Reuse

BARNETT, J. R.

- BATO88C: Construction of File Management Systems from Software Components

BARSTOW, DAVID R.

- BARS85A: Domain-Specific Automatic Programming

BATORY, DON S.

- BATO88A: Building Blocks of Database Management Systems
- BATO88B: Concepts for a Database System Compiler
- BATO88C: Construction of File Management Systems from Software Components

BENNETT, JAMES S.

- BENN84A: ROGET: Acquiring the Conceptual Structure of a Diagnostic Expert System

BIGGERSTAFF, TED J.

- BIGG88B: The Nature of Semi-Formal Information in Domain Models

BORGIDA, ALEXANDER

- BORG84A: Generalization/Specialization as a Basis for Software Specifications
- BORG85A: Features of Languages for the Development of Information Systems at the Conceptual Level

BROWNE, JAMES C.

- ISCO89A: Generating Domain Models for Program Specification and Generation

BRUNS, GLENN

- BRUN88A: Domain Modeling Approaches to Software Development

BURLON, RICCARDO

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

CARDILE, BRUNO

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

CARLE, RICK

- CARL87A: Reusable Software Components for Missile Applications

CHAND, D. R.

- YADA89A: An expert modeling support system for modeling an object system to specify its information requirements

CLEMENTS, PAUL C.

- PARN85A: The Modular Structure of Complex Systems

COHEN, JOEL

- COHE89A: Software Reuse for Information Management Systems

COHEN, SHOLOM G.

- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System

COMER, EDWARD R.

- GILR89A: Impact of Domain Analysis on Reuse Methods

CONTI, MIRELLA

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

D'IPPOLITO, RICHARD S.

- DIPP89B: Using Models in Software Engineering

DIETZEN, SCOTT R.

- DIET87A: Analogy in Program Development

DOWLE, S. W.

- DOWL89A: Domain Modelling for Requirements Specification

DUBOIS, ERIC

- DUBO86A: A Knowledge Representation Language for Requirements Engineering

DUREK, THOMAS A.

- JAWO90A: A Domain Analysis Process

FAULK, STUART

- JAWO90A: A Domain Analysis Process

FICKAS, STEPHEN

- FICK87B: Automating the Specification Process

FINKELSTEIN, ANTHONY

- FINK88A: Re-use of Formatted Requirements Specifications

FISHER, GARY LEE

- FISH86A: A Software Architecture for Strategic Management Support Systems Utilizing Interactive Graphics

FREEMAN, PETER

- FREE87A: A Conceptual Analysis of the Draco Approach to Constructing Software Systems

GAFFNEY, JOHN E.

- JAWO90A: A Domain Analysis Process

GARZA, JORGE F.

- BATO88C: Construction of File Management Systems from Software Components

GIDDINGS, RICHARD V.

- GIDD84A: Accommodating Uncertainty in Software Design

GILROY, KATHLEEN A.

- GILR89A: Impact of Domain Analysis on Reuse Methods

GOMAA, HASSAN

- GOMA90A: A Domain Requirements Analysis and Specification Method

GOODELL, MICHAEL

- GOOD83A: Quantitative Study of Functional Commonality in a Sample of Commercial Business Applications

GRAU, J. KAYE

- GILR89A: Impact of Domain Analysis on Reuse Methods

GREENSPAN, SOL J.

- GREE88A: Toward an Object-Oriented Framework for Defining Services in Future Intelligent Networks

GUERRIERI, ERNESTO

- VITA90B: Domain Analysis within the ISEC Rapid Center

HARANDI, MEHDI T.

- HARA89A: Automating Software Specification and Design

HILLS, FRED

- JAWO90A: A Domain Analysis Process

HINDLEY, P. G.

- HUTC88A: A Preliminary Study of Large-Scale Software Re-use

HOLTZMAN, PETER L.

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

HRYCEJ, TOMAS

- HRYC87A: A Knowledge-Based Problem-Specific Program Generator

HUTCHINSON, J. W.

- HUTC88A: A Preliminary Study of Large-Scale Software Re-use

ISCOE, NEIL

- ISCO88A: Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach
- ISCO89A: Generating Domain Models for Program Specification and Generation

JAWORSKI, ALLAN

- JAWO90A: A Domain Analysis Process

KANG, KYO C.

- KANG89A: Results from Domain Analysis Working Group
- KANG89B: Features Analysis: An Approach to Domain Analysis

LATOUR, LARRY

- LATO89B: Issues Involved in the Content and Organization of Software Component Information Bases: Interim Report

LEDBETTER, LAMAR

- LEDB83A: Reusability of Domain Knowledge in the Automatic Programming System (phi)

LEE, KENNETH J.

- LEE88A: An OOD Paradigm for Flight Simulators, 2nd Edition

LUBARS, MITCHELL D.

- HARA89A: Automating Software Specification and Design
- LUBA87A: A Knowledge-based Design Aid for the Construction of Software Systems
- LUBA88A: A Domain Modeling Representation
- LUBA88B: Domain Analysis and Domain Engineering in IDeA

MANO, NOBUOKI

- MANO88A: Modeling of Data-Processing Software for Generating and Reusing Their Programs

MAURI, JOHN

- MAUR88A: Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design

MCGOWAN, C. L.

- GREE88A: Toward an Object-Oriented Framework for Defining Services in Future Intelligent Networks

MCLEOD, MARK L.

- MAUR88A: Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design

MCNICHOLL, DANIEL G.

- MCNI86A: Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results
- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System

MEHROTRA, PREM K.

- ROSA88A: MES: An Expert System for Reusing Models of Transmission Equipment

MELLOR, STEPHEN J.

- SHLA89A: An Object-Oriented Approach to Domain Analysis
- SHLA90A: Recursive Design

MERLET, PATRICK J.

- GILR89A: Impact of Domain Analysis on Reuse Methods

MOORE, JOHN M.

- MOOR88A: Position Paper on Domain Analysis
- MOOR89A: Domain Analysis: Framework for Reuse

MOSTOW, JOHN

- MOST85A: Toward Better Models of the Design Process

MYERS, BRAD A.

- MYER88A: A Taxonomy of Window Manager User Interfaces

MYLOPOULOS, JOHN

- BORG84A: Generalization/Specialization as a Basis for Software Specifications

NEIGHBORS, JAMES M.

- NEIG80A: Software Construction Using Components
- NEIG83A: The Draco Approach to Constructing Software from Reusable Components
- NEIG87A: Report on the Domain Analysis Working Group Session

PALMER, CONSTANCE

- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume 1: CAMP Parts and Parts Composition System

PARNAS, DAVID L.

- PARN76A: On the Design and Development of Program Families
- PARN79A: Designing Software for Ease of Extension and Contraction
- PARN85A: The Modular Structure of Complex Systems

PAYTON, TERI F.

- PAYT88B: Reusability Library Framework

PERRY, JAMES M.

- PERR89A: The Role of Domain Independence in Promoting Software Reuse: Architectural Analysis of Systems

PIETRI, FRANCESCO

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

POLLOCK, GUYLAINE MARIE

- POLL85A: A Design Methodology and Support Environment for Complexity Metrics via Reusable Software Parts

POTTS, COLIN

- BRUN88A: Domain Modeling Approaches to Software Development

PRIETO-DIAZ, RUBEN

- PRIE87C: Domain Analysis for Reusability
- PRIE87D: Faceted Classification and Reuse Across Domains
- PRIE89A: Domain Analysis Tutorial

PUNCELLO, P. PAOLO

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

ROBINSON, WILLIAM N.

- ROBI89B: Integrating Multiple Specifications Using Domain Goals

ROSALES, S. ROODY

- ROSA88A: MES: An Expert System for Reusing Models of Transmission Equipment

ROY, J.

- BATO88C: Construction of File Management Systems from Software Components

SCHERLIS, WILLIAM L.

- DIET87A: Analogy in Program Development
- SCHE86A: Abstract Data Types, Specialization, and Program Reuse

SHAW, MARY

- PERR89A: The Role of Domain Independence in Promoting Software Reuse: Architectural Analysis of Systems

SHEKERAN, M. C.

- GREE88A: Toward an Object-Oriented Framework for Defining Services in Future Intelligent Networks

SHLAER, SALLY

- SHLA89A: An Object-Oriented Approach to Domain Analysis
- SHLA90A: Recursive Design

SIDORAN, JAMES L.

- SIDO89A: Conceptual Modeling and C3I System Design

SIMOS, MARK A.

- SIMO87A: The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability
- SIMO87B: The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability

SOLDERITSCH, JAMES J.

- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries

SOLOWAY, ELLIOT

- ADEL85A: The Role of Domain Experience in Software Design

TERATSUJI, EIICHI

- ARAN88C: Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse

THALHAMER, JOHN A.

- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries

TORRIGIANI, PIERO

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

TWICHELL, B. C.

- BATO88C: Construction of File Management Systems from Software Components

UNISYS

- UNIS88A: Reusability Library Framework AdaKNET/AdaTAU Design Report

VITALETTI, WILLIAM

- VITA90B: Domain Analysis within the ISEC Rapid Center

WALLNAU, KURT C.

- SOLD89A: Constructing Domain Specific Ada Reuse Libraries

WARD, PAUL T.

- WARD88A: Using the Structured Techniques to Support Software Reuse

WEBSTER, DALLAS E.

- WEBS88A: Mapping the Design Information Representation Terrain

WEISS, DAVID M.

- PARN85A: The Modular Structure of Complex Systems
- WEIS88A: Reuse and Prototyping: A Methodology

WERTH, JOHN

- ISCO89A: Generating Domain Models for Program Specification and Generation

WILLIAMS, LLOYD G.

- WARD88A: Using the Structured Techniques to Support Software Reuse

WONG, H.

- BORG84A: Generalization/Specialization as a Basis for Software Specifications

YADAV, SURYA B.

- YADA88A: Comparison of Analysis Techniques for Information Requirement Determination
- YADA89A: An expert modeling support system for modeling an object system to specify its information requirements

Appendix II
Chronological

1976

- PARN76A: On the Design and Development of Program Families

1979

- PARN79A: Designing Software for Ease of Extension and Contraction

1980

- NEIG80A: Software Construction Using Components

1983

- GOOD83A: Quantitative Study of Functional Commonality in a Sample of Commercial Business Applications
- LEDB83A: Reusability of Domain Knowledge in the Automatic Programming System (phi)
- NEIG83A: The Draco Approach to Constructing Software from Reusable Components

1984

- BENN84A: ROGET· Acquiring the Conceptual Structure of a Diagnostic Expert System
- BORG84A: Generalization/Specialization as a Basis for Software Specifications
- GIDD84A: Accommodating Uncertainty in Software Design

1985

- ADEL85A: The Role of Domain Experience in Software Design
- BARS85A: Domain-Specific Automatic Programming
- BORG85A: Features of Languages for the Development of Information Systems at the Conceptual Level
- MOST85A: Toward Better Models of the Design Process
- PARN85A: The Modular Structure of Complex Systems
- POLL85A: A Design Methodology and Support Environment for Complexity Metrics via Reusable Software Parts

1986

- ALEX86A: Knowledge Level Engineering: Ontological Analysis
- DUBO86A: A Knowledge Representation Language for Requirements Engineering
- FISH86A: A Software Architecture for Strategic Management Support Systems Utilizing Interactive Graphics
- MCNI86A: Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results
- SCHE86A: Abstract Data Types, Specialization, and Program Reuse

1987

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development

Workstation Project

- CARL87A: Reusable Software Components for Missile Applications
- DIET87A: Analogy in Program Development
- FREE87A: A Conceptual Analysis of the Draco Approach to Constructing Software Systems
- HRYC87A: A Knowledge-Based Problem-Specific Program Generator
- LUBA87A: A Knowledge-based Design Aid for the Construction of Software Systems
- NEIG87A: Report on the Domain Analysis Working Group Session
- PRIE87C: Domain Analysis for Reusability
- PRIE87D: Faceted Classification and Reuse Across Domains
- SIMO87A: The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability
- SIMO87B: The Domain-Oriented Software Life Cycle: Towards an Extended Process Model for Reusability

1988

- ARAN88A: Domain Engineering for Software Reuse
- ARAN88B: Evaluation of a Reuse-Based Software Construction Technology
- ARAN88C: Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse
- ASDJ88A: Knowledge-Based Component Composition: An Approach to Software Reusability
- BAIL88B: Semi-Automatic Development of Payload Operations Control Center Software
- BATO88A: Building Blocks of Database Management Systems
- BATO88B: Concepts for a Database System Compiler
- BATO88C: Construction of File Management Systems from Software Components
- BIGG88B: The Nature of Semi-Formal Information in Domain Models
- BRUN88A: Domain Modeling Approaches to Software Development
- FINK88A: Re-use of Formatted Requirements Specifications
- GREE88A: Toward an Object-Oriented Framework for Defining Services in Future Intelligent Networks
- HUTC88A: A Preliminary Study of Large-Scale Software Re-use
- ISCO88A: Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach
- LEE88A: An OOD Paradigm for Flight Simulators, 2nd Edition
- LUBA88A: A Domain Modeling Representation
- LUBA88B: Domain Analysis and Domain Engineering in IDeA

- MANO88A: Modeling of Data-Processing Software for Generating and Reusing Their Programs
- MAUR88A: Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design
- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System
- MOOR88A: Position Paper on Domain Analysis
- MYER88A: A Taxonomy of Window Manager User Interfaces
- PAYT88B: Reusability Library Framework
- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)
- ROSA88A: MES: An Expert System for Reusing Models of Transmission Equipment
- UNIS88A: Reusability Library Framework AdaKNET/AdaTAU Design Report
- WARD88A: Using the Structured Techniques to Support Software Reuse
- WEBS88A: Mapping the Design Information Representation Terrain
- WEIS88A: Reuse and Prototyping: A Methodology
- YADA88A: Comparison of Analysis Techniques for Information Requirement Determination

1989

- ARAN89A: Domain Analysis - From Art Form to Engineering Discipline
- BAIL89C: Generic POCC Architectures
- BAIL89D: The KAPTUR Environment: An Operations Concept
- COHE89A: Software Reuse for Information Management Systems
- DIPP89B: Using Models in Software Engineering
- DOWL89A: Domain Modelling for Requirements Specification
- FICK87B: Automating the Specification Process
- GILR89A: Impact of Domain Analysis on Reuse Methods
- HARA89A: Automating Software Specification and Design
- ISCO89A: Generating Domain Models for Program Specification and Generation
- KANG89A: Results from Domain Analysis Working Group
- KANG89B: Features Analysis: An Approach to Domain Analysis
- LATO89B: Issues Involved in the Content and Organization of Software Component Information Bases: Interim Report
- MOOR89A: Domain Analysis: Framework for Reuse
- PERR89A: The Role of Domain Independence in Promoting Software Reuse: Architectural Analysis of Systems
- PRIE89A: Domain Analysis Tutorial

- ROBI89B: Integrating Multiple Specifications Using Domain Goals
- SHLA89A: An Object-Oriented Approach to Domain Analysis
- SIDO89A: Conceptual Modeling and C3I System Design
- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries
- YADA89A: An expert modeling support system for modeling an object system to specify its information requirements

1990

- GOMA90A: A Domain Requirements Analysis and Specification Method
- JAWO90A: A Domain Analysis Process
- SHLA90A: Recursive Design
- VITA90B: Domain Analysis within the ISEC Rapid Center

Appendix III
Alphabetical by Organization/Project

BAUHAUS

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

CAMP

- MCNI86A: Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results
- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System
- PRIE89A: Domain Analysis Tutorial

CMU

- MYER88A: A Taxonomy of Window Manager User Interfaces

CTA

- BAIL88B: Semi-Automatic Development of Payload Operations Control Center Software
- BAIL89C: Generic POCC Architectures
- BAIL89D: The KAPTUR Environment: An Operations Concept
- MOOR88A: Position Paper on Domain Analysis
- MOOR89A: Domain Analysis: Framework for Reuse

DESIRE

- BIGG88B: The Nature of Semi-Formal Information in Domain Models

DRACO

- ARAN88B: Evaluation of a Reuse-Based Software Construction Technology
- BRUN88A: Domain Modeling Approaches to Software Development
- FREE87A: A Conceptual Analysis of the Draco Approach to Constructing Software Systems
- NEIG80A: Software Construction Using Components
- NEIG83A: The Draco Approach to Constructing Software from Reusable Components

ESPRIT

- PUNC88A: ASPIS: a knowledge-based CASE environment. (Application Software Prototype Implementation System)

GENESIS

- BATO88A: Building Blocks of Database Management Systems
- BATO88B: Concepts for a Database System Compiler
- BATO88C: Construction of File Management Systems from Software Components

GIST

- BRUN88A: Domain Modeling Approaches to Software Development

GTE

- COHE89A: Software Reuse for Information Management Systems
- PRIE87C: Domain Analysis for Reusability
- PRIE87D: Faceted Classification and Reuse Across Domains
- PRIE89A: Domain Analysis Tutorial

IDEA

- HARA89A: Automating Software Specification and Design
- LUBA87A: A Knowledge-based Design Aid for the Construction of Software Systems
- LUBA88B: Domain Analysis and Domain Engineering in IDeA

INFERENCE

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

KAPTUR

- BAIL89D: The KAPTUR Environment: An Operations Concept

KATE

- FICK87B: Automating the Specification Process
- PRIE89A: Domain Analysis Tutorial

MCC

- BIGG88B: The Nature of Semi-Formal Information in Domain Models
- BRUN88A: Domain Modeling Approaches to Software Development
- HARA89A: Automating Software Specification and Design
- LUBA88A: A Domain Modeling Representation
- LUBA88B: Domain Analysis and Domain Engineering in IDeA
- WEBS88A: Mapping the Design Information Representation Terrain

MCDONNELL-DOUGLAS

- MCNI86A: Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results
- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System

NASA

- ALLE87A: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project
- BAIL88B: Semi-Automatic Development of Payload Operations Control Center Software
- BAIL89C: Generic POCC Architectures
- BAIL89D: The KAPTUR Environment: An Operations Concept
- MOOR88A: Position Paper on Domain Analysis
- MOOR89A: Domain Analysis: Framework for Reuse

OZYM

- ISCO89A: Generating Domain Models for Program Specification and Generation

PHINIX

- BARS85A: Domain-Specific Automatic Programming
- LEDB83A: Reusability of Domain Knowledge in the Automatic Programming System (phi)

RADC

- MAUR88A: Cartographic Applications for Tactical and Strategic Systems (CATSS) Functional Software Design

RAYTHEON

- CARL87A: Reusable Software Components for Missile Applications

RLF

- PAYT88B: Reusability Library Framework
- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries
- UNIS88A: Reusability Library Framework AdaKNET/AdaTAU Design Report

ROGET

- BENN84A: ROGET: Acquiring the Conceptual Structure of a Diagnostic Expert System

SCHLUMBERGER-DOLL

- BARS85A: Domain-Specific Automatic Programming
- LEDB83A: Reusability of Domain Knowledge in the Automatic Programming System (phi)

SEI

- DIPP89B: Using Models in Software Engineering
- KANG89A: Results from Domain Analysis Working Group
- KANG89B: Features Analysis: An Approach to Domain Analysis
- LEE88A: An OOD Paradigm for Flight Simulators, 2nd Edition
- PERR89A: The Role of Domain Independence in Promoting Software Reuse: Architectural Analysis of Systems

SPC

- JAWO90A: A Domain Analysis Process
- WEIS88A: Reuse and Prototyping: A Methodology

SPS

- GILR89A: Impact of Domain Analysis on Reuse Methods

STARS

- MCNI86A: Common Ada Missile Packages (CAMP) - Volume I: Overview and Commonality Study Results
- MCNI88A: Common Ada Missile Packages - Phase 2 (CAMP-2) - Volume I: CAMP Parts and Parts Composition System
- PAYT88B: Reusability Library Framework

- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries
- UNIS88A: Reusability Library Framework AdaKNET/AdaTAU Design Report

U-ARIZONA

- FISH86A: A Software Architecture for Strategic Management Support Systems Utilizing Interactive Graphics

U-MAINE

- LATO89B: Issues Involved in the Content and Organization of Software Component Information Bases: Interim Report

U-OREGON

- FICK87B: Automating the Specification Process
- ROBI89B: Integrating Multiple Specifications Using Domain Goals

UA-HUNTSVILLE

- ASDJ88A: Knowledge-Based Component Composition: An Approach to Software Reusability

UC-IRVINE

- ARAN88A: Domain Engineering for Software Reuse
- ARAN88B: Evaluation of a Reuse-Based Software Construction Technology
- ARAN88C: Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse
- ARAN89A: Domain Analysis - From Art Form to Engineering Discipline
- FREE87A: A Conceptual Analysis of the Draco Approach to Constructing Software Systems
- NEIG80A: Software Construction Using Components
- NEIG83A: The Draco Approach to Constructing Software from Reusable Components

UI-URBANA

- HARA89A: Automating Software Specification and Design
- LUBA87A: A Knowledge-based Design Aid for the Construction of Software Systems

UNISYS

- PAYT88B: Reusability Library Framework
- SOLD89A: Constructing Domain-Specific Ada Reuse Libraries
- UNIS88A: Reusability Library Framework AdaKNET/AdaTAU Design Report

USA-CECOM

- GILR89A: Impact of Domain Analysis on Reuse Methods
- LATO89B: Issues Involved in the Content and Organization of Software Component Information Bases: Interim Report

UT-AUSTIN

- BATO88A: Building Blocks of Database Management Systems

- BATO88B: Concepts for a Database System Compiler
- BATO88C: Construction of File Management Systems from Software Components
- ISCO88A: Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach
- ISCO89A: Generating Domain Models for Program Specification and Generation

Index

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-90-SR-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INST. / 6b. OFFICE SYMBOL (If applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS HANSCOM AIR FORCE BASE, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE / 8b. OFFICE SYMBOL (If applicable) ESD/ AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	10 SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 63752F / PROJECT NO.: N/A / TASK NO.: N/A / WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) A DOMAIN ANALYSIS BIBLIOGRAPHY	
12. PERSONAL AUTHOR(S) James A. Hess, William E. Novak, Patrick C. Carroll, Sholom G. Cohen, et al	
13a. TYPE OF REPORT FINAL / 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) June 1990 / 15. PAGE COUNT 56
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB GR	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) bibliography domain analysis software reuse

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document presents a bibliography of references on a comparatively new discipline called domain analysis. This discipline defines a process to identify and represent the relevant information in a domain (a set of systems which share common capabilities). The information is derived from:

1. the study of existing systems and their development histories
2. knowledge captured from domain experts
3. underlying theory
4. emerging technology

Domain analysis has received considerable attention since the early 1980s. This interest stems from the fact that the application of domain analysis is now believed to be part of the foundation upon which a successful and systematic program of software reuse can be built. This foundation is achieved by capturing and preserving the information to be

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ OTIC USERS ☒	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED DISTRIBUTION
22a. NAME OF RESPONSIBLE INDIVIDUAL JOHN S. HERMAN, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) 412 268-7630 / 22c. OFFICE SYMBOL ESD/AVS (SEI JPO)

reused in future developments in the form of application-specific tools and reusable software models, architectures, and components.

This bibliography has been compiled as a part of the work on the Domain Analysis Project at the Software Engineering Institute. The bibliography's purpose is to provide an historical perspective on the field as well as a necessary background for further work in the discipline.