

Technical Report  
CMU/SEI-95-TR-019  
ESC-TR-95-019

**An Evolutionary Perspective of Software Engineering  
Research Through Co-Word Analysis**

Neal Coulter  
Ira Monarch  
Suresh Konda  
Marvin Carr

March 1996

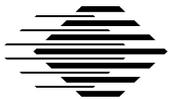
**Technical Report**

CMU/SEI-95-TR-019

ESC-TR-95-019

March 1996

An Evolutionary Perspective of Software Engineering  
Research Through Co-Word Analysis



Neal Coulter

Florida Atlantic University

Ira Monarch

Suresh Konda

Marvin Carr

Software Engineering Institute

Risk Program

Unlimited distribution subject to the copyright.

**Software Engineering Institute**

Carnegie Mellon University  
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office  
HQ ESC/ENS  
5 Eglin Street  
Hanscom AFB, MA 01731-2116

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.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF  
SEI Joint Program Office

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

Copyright © 5/10/96 by Carnegie Mellon University.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

Requests for permission to reproduce this document or to prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is <http://www.rai.com>

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation for This Empirical Study	1
1.2	Questions Addressed, Tools Employed	1
1.3	Intended Audiences –Alternative Routes Through the Paper	3
<b>2</b>	<b>The Data and Its Descriptors</b>	<b>5</b>
<b>3</b>	<b>The Metric and the Algorithm</b>	<b>9</b>
3.1	The Metric	9
3.2	The Algorithm	9
3.2.1	Pass-1	10
3.2.2	Pass-2	12
3.2.3	Algorithm Constraints	15
3.2.4	Algorithm Summary	15
3.3	Comments on Selection of Network Parameters	16
<b>4</b>	<b>Network Analysis</b>	<b>17</b>
4.1	Network Names	17
4.1.1	Methodology	17
4.1.2	Findings	17
4.1.2.1	1982-1986 Networks	17
4.1.2.2	1987-1990 Networks	18
4.1.2.3	1991-1994 Networks	18
4.2	Network Summaries	21
4.2.1	Methodology	21
4.2.2	Findings	21
<b>5</b>	<b>Types of Networks and Their Interactions</b>	<b>25</b>
5.1	Methodology	25
5.2	Findings	25
5.3	Evidence of a Coalescing Field	27
<b>6</b>	<b>Super Network Analysis</b>	<b>29</b>
6.1	Methodology	29
6.2	Findings	29
<b>7</b>	<b>Trends over Periods</b>	<b>35</b>
7.1	Analysis of Descriptor Contexts	35
7.2	Analysis of Networks Across Time Periods	39

7.2.1	Methodology	39
7.2.2	Findings	39
7.2.3	Similarity Index Within a Time Period	42
<b>8</b>	<b>Descriptor Analysis</b>	<b>43</b>
8.1	Analysis	43
8.2	Findings	47
<b>9</b>	<b>Conclusions</b>	<b>49</b>
9.1	Methodology	49
9.2	Findings	49
9.2.1	The Role of Software Tools	49
9.2.2	Software Engineering and Computer Science	50
9.2.3	Limitations of This Study	50
<b>10</b>	<b>References</b>	<b>51</b>
	<b>Appendix: Maps of All Networks</b>	<b>55</b>
A.1	1982-1986 Maps of 15 Networks	56
A.2	1987-1990 Maps of 16 Networks	64
A.3	1991-1994 Maps of 11 Networks	72

# List of Figures

<b>Figure 1:</b>	First Example of a Pass-1 Network	11
<b>Figure 2:</b>	Second Example of a Pass-1 Network	11
<b>Figure 3:</b>	Pass-1 and Pass-2 Nodes and Links	13
<b>Figure 4:</b>	1982-1986 Centrality and Density	26
<b>Figure 5:</b>	1987-1990 Centrality and Density	26
<b>Figure 6:</b>	1991-1994 Centrality and Density	27
<b>Figure 7:</b>	1982-1986 Super Network, = 2	33
<b>Figure 8:</b>	1987-1990 Super Network, = 2	33
<b>Figure 9:</b>	1991-1994 Super Network, = 2	34
<b>Figure 10:</b>	Ada in Network-1, 1982-1986	36
<b>Figure 11:</b>	Ada in Network-10, 1987-1990	36
<b>Figure 12:</b>	Ada in Network-7, 1991-1994	37
<b>Figure 13:</b>	Structured Programming in Network-5, 1982-1986	38
<b>Figure 14:</b>	Structured Programming in Network-8, 1987-1990	38
<b>Figure A.1-1:</b>	Software Management - Ada	56
<b>Figure A.1-2:</b>	Logic Programming	56
<b>Figure A.1-3:</b>	User Interfaces	57
<b>Figure A.1-4:</b>	Standards	57
<b>Figure A.1-5:</b>	Tools and Techniques - Structured Programming - Pascal	58
<b>Figure A.1-6:</b>	Software Development	58
<b>Figure A.1-7:</b>	Software Libraries	59
<b>Figure A.1-8:</b>	Testing and Debugging - Correctness Proofs	59
<b>Figure A.1-9:</b>	Reliability	60
<b>Figure A.1-10:</b>	Program Editors	60
<b>Figure A.1-11:</b>	Requirements/Specifications - Systems Analysis and Design	61
<b>Figure A.1-12:</b>	Modules and Interfaces	61
<b>Figure A.1-13:</b>	Real-Time Systems	62
<b>Figure A.1-14:</b>	Abstract Data Types	62
<b>Figure A.1-15:</b>	Metrics	63
<b>Figure A.2-1:</b>	Geometrical Problems and Computations	64
<b>Figure A.2-2:</b>	Correctness Proofs - Languages	64
<b>Figure A.2-3:</b>	Logic Programming	65
<b>Figure A.2-4:</b>	Requirements/Specifications - Methodologies	65
<b>Figure A.2-5:</b>	User/Machine Systems	66

<b>Figure A.2-6:</b>	Methodologies - Software Development	66
<b>Figure A.2-7:</b>	Standards	67
<b>Figure A.2-8:</b>	Structured Programming	67
<b>Figure A.2-9:</b>	Applications and Expert Systems - Tools and Techniques	68
<b>Figure A.2-10:</b>	Concurrent Programming - Ada	68
<b>Figure A.2-11:</b>	Computer-Aided Design	69
<b>Figure A.2-12:</b>	Error Handling and Recovery	69
<b>Figure A.2-13:</b>	Distribution and Maintenance	70
<b>Figure A.2-14:</b>	Software Configuration Management	70
<b>Figure A.2-15:</b>	Reusable Software	71
<b>Figure A.2-16:</b>	Software Management - Design	71
<b>Figure A.3-1:</b>	User Interfaces	72
<b>Figure A.3-2:</b>	Petri Nets	72
<b>Figure A.3-3:</b>	Software Development - Object-Oriented Programming	73
<b>Figure A.3-4:</b>	Software Libraries - C++ - Microsoft Windows	73
<b>Figure A.3-5:</b>	Windows	74
<b>Figure A.3-6:</b>	X-Windows	74
<b>Figure A.3-7:</b>	Tools and Techniques - CASE - Systems Analysis and Design	75
<b>Figure A.3-8:</b>	Requirements/Specifications	75
<b>Figure A.3-9:</b>	General	76
<b>Figure A.3-10:</b>	Concurrent Programming	76
<b>Figure A.3-11:</b>	Metrics	77

## List of Tables

<b>Table 1:</b>	Distribution of Documents by Year	7
<b>Table 2:</b>	Documents and Descriptors per Time Period	7
<b>Table 3:</b>	Links in Decreasing Order of Strength	14
<b>Table 4:</b>	Parameters and Resulting Networks	15
<b>Table 5:</b>	Network Names and Numbers	20
<b>Table 6:</b>	1982-1986 Network Summary Data	22
<b>Table 7:</b>	1987-1990 Network Summary Data	23
<b>Table 8:</b>	1991-1994 Network Summary Data	24
<b>Table 9:</b>	Comparison of Properties for Time Periods	28
<b>Table 10:</b>	Possible 1982-1986 Super Networks	30
<b>Table 11:</b>	Possible 1991-1994 Super Networks	31
<b>Table 12:</b>	Possible 1987-1990 Super Networks	32
<b>Table 13:</b>	Summary of Descriptor Data	44
<b>Table 14:</b>	CCS Descriptor Summary Data	46

## Acknowledgments

David Gluch, Nancy Mead, Robert Park, Will Hayes, Eswaran Subrahmanian, Mario Barbacci, Anthony Ralston, Christopher Fox, and Dennis Frailey provided many helpful suggestions for improving the content and clarity of this paper. We thank Ronald Higuera and Clyde Chittister for their support in this extended study. Suzanne Couturiaux and Tanya Jones greatly assisted in final preparation of the report.

# An Evolutionary Perspective of Software Engineering Research Through Co-Word Analysis

**Abstract:** This study applies various tools, techniques, and methods that the Software Engineering Institute is evaluating for analyzing information being produced at a very rapid rate in the discipline—both in practice and in research. The focus here is on mapping the evolution of the research literature as a means to characterize software engineering and distinguish it from other disciplines. Software engineering is a term often used to describe programming-in-the-large activities. Yet, any precise empirical characterization of its conceptual contours and their evolution is lacking. In this study, a large number of publications from 1982-1994 are analyzed to determine themes and trends in software engineering.

The method used to analyze the publications was co-word analysis. This methodology identifies associations among publication descriptors (indexing terms) from the Computing Classification System and produces networks of terms that reveal patterns of associations. The results suggest that certain research themes in software engineering remain constant, but with changing thrusts. Other themes mature and then diminish as major research topics, while still others seem transient or immature. Certain themes are emerging as predominate for the most recent time period covered (1991-1994): object-oriented methods and user interfaces are identifiable as central themes.

## 1 Introduction

### 1.1 Motivation for This Empirical Study

Engineering disciplines in both research and practice generate information at a very rapid rate, and software engineering is no exception. The Software Engineering Institute is evaluating various tools, techniques, and methods that aid in managing this information explosion for the discipline of software engineering, the Software Engineering Institute itself, as well as organizations doing software-intensive work. This study focuses on the discipline of software engineering as a whole, especially with respect to research literature being produced in the field. It is important to note, however, that many of the same tools, techniques, and methods useful for filtering information and for detecting patterns and trends at the global research level are also applicable at the local organizational level.

### 1.2 Questions Addressed, Tools Employed

Interesting discussions about the nature and status of software engineering have occurred in recent years. We thought it would be interesting to explore this issue by letting the research in software engineering describe itself through the medium of the information management tools we had been investigating. We formulated the questions as the following:

Is software engineering a child of computer science, computer engineering, or information systems, or is it an intersecting—but relatively independent—discipline? Is software engineering changing with respect to its primary foci?

These are important issues in industry and academe because they address research, application, and curriculum concerns. These topics have been discussed by many professionals [Ford 89], [Denning 92], [Dijkstra 89], [Gibbs 91], [Gibbs 89], [Gries 91], [Parnas 90], [Parnas 85], [Denning 89], [Shaw 90], [Jackson 94], [Brooks 87], [Coulter 94]. In addition, a special ACM/IEEE Computer Society task force is now commissioned to consider the matter [Buckley 93], [Boehm 94].

While discussions about computer science/software engineering are useful, empirical studies of the issue are also needed. Such studies require a carefully considered methodology and accompanying data sets. The methodology we have chosen is based on co-word analysis [Callon 86], [Callon 91], [Courtial 89], [Law 92], [Whittaker 89]. Co-word analysis reveals patterns of associations among terms by measuring and representing the associations of terms describing technical publications or other technical texts.

This study uses co-word analysis to provide insight into the nature of software engineering. Our hypothesis is that the identified patterns of term associations are maps of the conceptual space of software engineering and its relations to other computing fields. Further, a series of such maps constructed for different time periods suggests a trace of the changes in this conceptual space.

The technique is applied to a very large cross-section of published text (1982-1994) in the computing field that is indexed with descriptors from the well-known Computing Classification System (CCS).<sup>1</sup> This indexed text comes from the Association for Computing Machinery's (ACM) *Guide to Computing Literature* (GUIDE), which covers an ACM publications database. Through professional indexers, GUIDE annually covers over 20,000 items by descriptors from the CCS.<sup>2</sup>

CCS is a carefully designed taxonomy that has existed since 1982 [Sammet 82], and it has been updated three times [Sammet 83], [Sammet 87], [Coulter 91]. Because CCS classifies publications over the breadth of computing, it allows us to investigate trends and the position of software engineering in the larger computing context.

---

1. Until 1996, the Computing Classification System (CCS) was called the Computing Reviews Classification System (CRCS).

2. Descriptors selected from CCS are distinguished from keywords freely chosen by the author. Only CCS descriptors were used in this study. The issue of descriptors selected by professional indexers, as opposed to free selection of keywords by the authors, is important here. While both may have merits, we believe it is useful to study a fixed system that imposes a common nomenclature across all computing. Professional indexers experienced in using the CCS assure standard application of that taxonomy. Law and Whittaker [Law 92], [Whittaker 89] addressed these issues extensively.

### 1.3 Intended Audiences –Alternative Routes Through the Paper

There are several different kinds of audiences for this paper. It can be read on three levels:

1. At one level it is an attempt to characterize software engineering as a discipline, both in its own right and in its important differences from other related disciplines. No particular background is required, though some familiarity with various issues in software engineering research or practice is necessary to appreciate the conclusions reached.
2. Some readers may be just as interested in finding out how useful the tools, techniques, and methods are in answering the kinds of questions posed by the study; they may have an interest in just how accurate and informative such approaches are at summarizing large amounts of information and detecting patterns and trends in it. A willingness to wade through some descriptions of information retrieval and statistical techniques is required, but these descriptions are self-contained.
3. A third group of readers might be interested in evaluating these tools, techniques, and methods to gain an understanding of how they might be applied in their own work. Here some familiarity with current work in information retrieval and computational linguistics would be useful, though not required, for anyone doing technical work in software engineering.

Following the introduction, a discussion of the data and its sources begins the main body of the paper. This includes the descriptors and codes used for indexing the software engineering documents, the sources of the documents indexed, and the numbers of documents covered in each of the respective time periods. This discussion of *what* is analyzed is followed by a discussion in the next section of *how* it is analyzed. In particular, the metric for determining co-occurrence strength between descriptors associated with the same documents is described. In the same section, the algorithm used to generate networks of co-occurring descriptors is detailed. Example networks generated from descriptors of the software engineering literature are presented.

Next comes a discussion of the methods used for interpreting networks: in particular, the method used for naming them and a more technical discussion of how complexity of networks is measured. These two discussions of methods are each followed by presentations of findings that list, analyze, and describe the networks found in the time periods covered. A more general discussion of types of networks comes next; it focuses on two of their distinguishing factors, called centrality and density. Examples from analyses of the current data are provided and implications discussed.

Methods for identifying relationships among networks within a time period are pursued next, followed by a discussion of what was found when these methods were applied to the networks generated. Then an analysis of the findings from each time period are compared and contrasted in order to determine how the discipline of software engineering has evolved over time. The evolution of the programming system Ada is presented as an example. Finally, the last section before the conclusion discusses the distributions of categories of descriptors from the point of view of those that made it into the co-occurrence networks against those that did not.

To assist the reader, some sections will have a heading for *Methodology* and for *Findings*. These sections will expand on the research methods and on the software engineering specifics, respectively.

## 2 The Data and Its Descriptors

Co-word methodology operates on indexed textual data. This chapter describes these two components for the study. Here, index terms used are taken directly from a standard taxonomy. In other similar applications at the SEI, software is used to generate index terms directly from the studied corpora.

GUIDE reviews and indexes a large number of publications across the spectrum of computing. Publications reviewed generally include books, book chapters, journals, proceedings, trade magazines and other applied sources, and occasionally other media such as videotaped material. For the latest list of publications received, see the November 1995 issue of *Computing Reviews* [CR 95]. In addition, GUIDE indexes many proceedings and articles from proceedings.

The CCS uses a four-level classification system. Any descriptors semantically below the fourth-level are nevertheless grouped at this level (note that all sections of the tree do not have four levels). The major CCS categories are listed below:

A-General Literature	G-Mathematics of Computing
B-Hardware	H-Information Systems
C-Computer Systems Organization	I-Computing Methodologies
D-Software	J-Computer Applications
E-Data	K-Computing Milieux
F-Theory of Computation	

The full CCS is described in the January 1996 issue of *Computing Reviews* [CR 96].

A complete rendition of the software engineering section of the taxonomy, D.2, follows. Superscripts indicate that a descriptor is new beginning with the year indicated;<sup>3</sup> all others are from the 1982 implementation.

---

<sup>3</sup> In some cases, a descriptor may appear in a document indexed before the official adoption of an updated version of CCS. Updates to CCS are always announced in January; however, the revision may be completed and in use by the indexers for a short time prior to its official release. Some documents written in one year may not be indexed until after a revision of CCS is in place, so the older version of CCS is no longer applied. In any case, these occurrences are not common.

D.2 SOFTWARE ENGINEERING	
D.2.0 General	D.2.6 Programming Environments
Protection mechanisms	Interactive <sup>87</sup>
Standards	D.2.7 Distribution and Maintenance
D.2.1 Requirements/Specifications	Corrections
Languages	Documentation
Methodologies	Enhancement
Tools	Extensibility
D.2.2 Tools and Techniques	Portability
Computer-aided software engineering (CASE) <sup>91</sup>	Restructuring
Decision table	Version control
Flow charts	D.2.8 Metrics
Modules and interfaces	Complexity measures
Petri nets <sup>91</sup>	Performance measures
Programmer workbench	Software science
Software libraries	D.2.9 Management
Structured programming	Copyrights
Top-down programming	Cost estimation
User interfaces	Life cycle
D.2.3 Coding	Productivity
Pretty printers	Programming teams
Program editors	Software configuration management
Reentrant code	Software quality assurance
Standards	Time estimation <sup>91</sup>
D.2.4 Program Verification	D.2.10 Design <sup>87</sup>
Assertion checkers	Methodologies <sup>87</sup>
Correctness proofs	Representation <sup>87</sup>
Reliability	D.2.m Miscellaneous
D.2.5 Testing and Debugging	Rapid prototyping <sup>83</sup>
Code inspections and walk-throughs <sup>91</sup>	Reusable software <sup>83</sup>
Debugging aids	
Diagnostics	
Dumps	
Error handling and recovery	
Symbolic execution	
Test data generators	
Tracing	

An item is almost always classified by multiple CCS descriptors. Even though there are up to four CCS levels, an item can be classified at any level that is appropriate; all branches of CCS do not have four levels. CCS does not include names of systems and languages (Unix, Ada, Windows etc.); instead, they are called *implicit subject descriptors* and can be used by indexers as needed. As we will see, their inclusion is common and often significant.

We obtained descriptors for all items indexed in GUIDE that had at least one descriptor in the D.2 category. Hence, the study admits descriptors from throughout CCS as long as an item has at least one D.2 descriptor. This selection allows us to examine interactions of software engineering nodes with other nodes in CCS. We could have refined this study by selecting more specific CCS descriptors (such as how Software Engineering [D.2] interacts with Programming Techniques [D.1],<sup>4</sup> for example). However, this study focuses on the larger question of how software engineering interacts with computing as a whole, i.e., on the interactions of software engineering with all other nodes of the CCS. The data we received reflect the March 1995 update to the GUIDE database.<sup>5</sup> Table 1 shows the numbers of indexed docu-

---

4. We show the corresponding CCS node after a descriptor for context when needed.

ments that we analyzed for the years 1982-1994.

**Table 1: Distribution of Documents by Year**

Year	Number of Documents
1982	81
1983	33
1984	211
1985	367
1986	1,027
1987	1,479
1988	2,329
1989	1,928
1990	1,914
1991	1,738
1992	2,016
1993	2,159
1994	1,612
Total	16,691

The total is 16,691 documents. As is evident, the number of documents was small until 1986. The 16,691 items were indexed by a total of 57,727 descriptors (a mean of 3.46 per item).

For analysis, we grouped the data for the years 1982-1986, 1987-1990, and 1991-1994. This separates the sparse years 1982-1986 from the others, gives approximately equal numbers of documents in the latter two periods, and provides breaks when CCS was updated so we do not confuse new descriptors across periods. Data for documents, descriptors, and their ratios for the time periods are shown in Table 2.

**Table 2: Documents and Descriptors per Time Period**

Time Period	Documents	Descriptors	Descriptor/Document Ratio
1982-1986	1,646	5,645	3.43
1987-1990	7,650	28,471	3.72
1991-1994	7,395	23,611	3.19

---

<sup>5</sup> Updates occur yearly, so this study is based on all available data. Note that updates do not necessarily reflect all of the entries from a single year. However, any demarcation is somewhat arbitrary, anyway, because of the nature of publication dates and subsequent inclusion in the database.



### 3 The Metric and the Algorithm

While some CCS-based results are presented here to demonstrate the methodology, this chapter focuses on underlying theory of co-word analysis. All readers need the material in this section.

Co-word analysis enables the structuring of data at various levels of analysis: (1) as *networks of links and nodes* (nodes in our networks contain *descriptors* that index documents.); (2) as distributions of *networks* called *super networks*; and (3) as transformations of networks and super networks over time periods. These structures and changing relationships provide a basis for tracing the evolution of software engineering.

Co-word analysis reduces a large space of related terms to multiple related smaller spaces that are easier to comprehend, but that also indicate actual partitions of interrelated concepts in the literature being analyzed. This analysis requires an association measure and an algorithm for searching through the space.

The analysis is designed to identify areas of strong focus that interrelate. This scheme allows us to construct a mosaic of software engineering topics.

#### 3.1 The Metric

Metrics for co-word analysis have been studied extensively [Callon 86], [Callon 91], [Courtial 89], [Law 92], [Whittaker 89]. The basic metric most suitable for this study is *Strength S* (called Equivalence Index by Callon). It is described as follows:

Two descriptors,  $i$  and  $j$ , co-occur if they are used together in the classification of a single document. Take a corpus consisting of  $N$  documents. Each document is indexed by a set of unique descriptors that can occur in multiple documents. Let  $c_k$  be the number of occurrences of descriptor  $k$ ; i.e, the number of times  $k$  is used for indexing documents in the corpus. Let  $c_{ij}$  be the number of co-occurrences of descriptors  $i$  and  $j$  (the number of documents indexed by both descriptor

Then Strength  $S$  of association between descriptors  $i$  and  $j$  is given by the expression:

$$S(c_i, c_j, c_{ij}) = \frac{c_{ij}^2}{c_i c_j}, 0 \leq S \leq 1.$$

Two descriptors that appear many times in isolation but only a few times together will yield a lower  $S$  value than two descriptors that appear relatively less often alone but have a higher ratio of co-occurrences.

#### 3.2 The Algorithm

The algorithm makes two passes through the data to produce pair-wise connections of descriptors in networks. A network consists of nodes (descriptors) connected by links. Each node must be linked to at least one other node in a network. The first pass (Pass-1) generates the primary associations among descriptors; these descriptors are called *internal nodes* and the corresponding links are called *internal links*. A second pass (Pass-2) generates links between

Pass-1 nodes across networks, thereby forming associations among completed networks. Pass-2 nodes and links are called *external* ones.

Pass-1 builds networks that can identify areas of strong focus; Pass-2 can identify descriptors that associate in more than one network and thereby indicate pervasive issues. This pattern of networks yields a mosaic of the data being analyzed.

### 3.2.1 Pass-1

During Pass-1, the link that has the highest strength is selected first. These linked nodes become the starting points for the first network. Other links and their corresponding nodes are then determined breadth-first.

Figure 1 illustrates this process for a 1991-1994 Pass-1 network. This figure displays the network connections as a map.<sup>6</sup> This network, named User Interfaces, is the first one created by the co-word algorithm for 1991-1994 data. The links are numbered in the order formed.

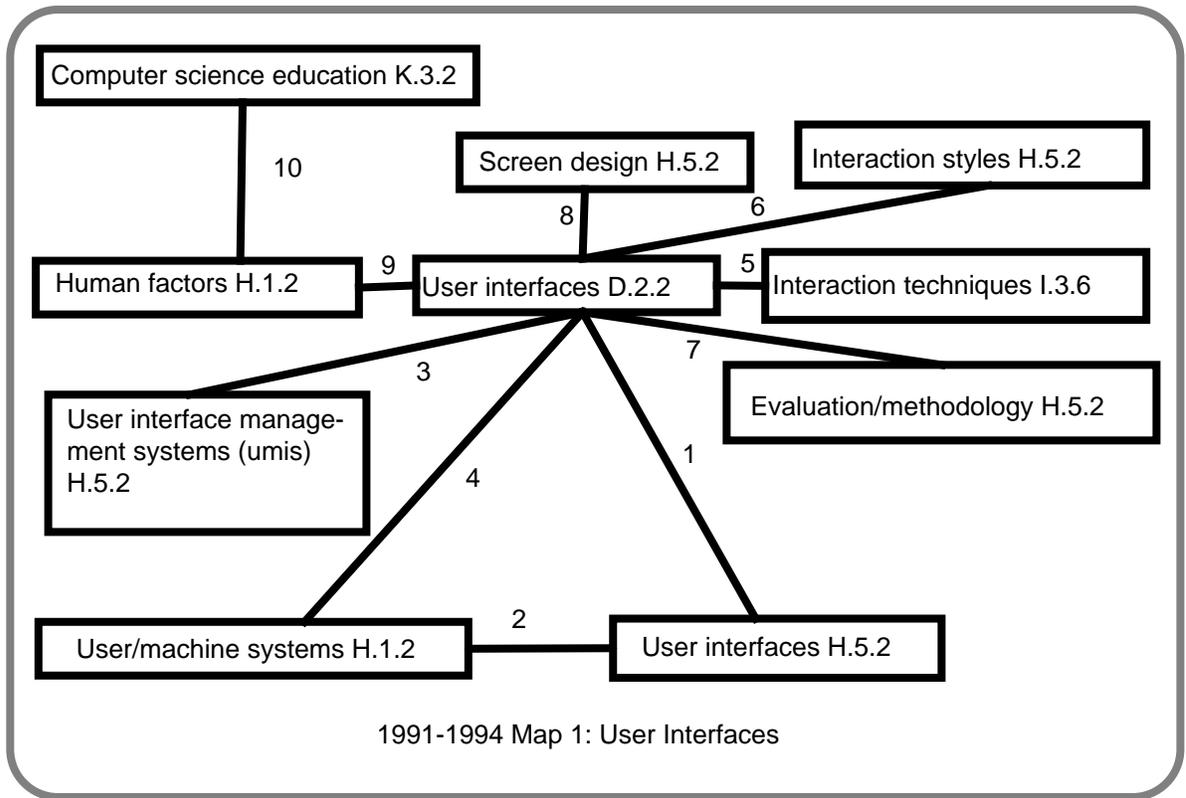
All nodes contained in the resulting Pass-1 network are removed from consideration for inclusion in subsequent Pass-1 networks. The next network then starts with the link of highest S value of the remaining links (i.e., ones not containing nodes from any previous network).

This Pass-1 strategy does not necessarily (or usually) yield S strengths in strict descending order, either within individual networks or among sequentially generated networks with respect to the sum or average of S strengths. The first network becomes the first network only because it starts with the highest link; the second network then starts with the highest link among remaining links, and so forth. This order of generation is not especially significant because it is possible that the links included in a network after the initial link do not have co-occurrence strengths in the same high range as this initial link.

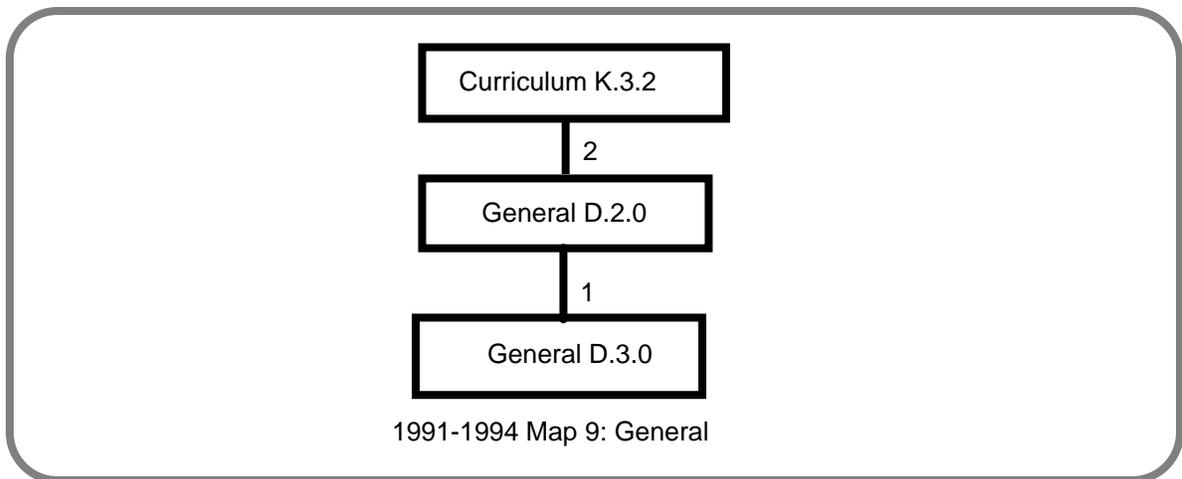
Figure 2 shows Pass-1 links for a second 1991-1994 network. This network, named General, was the ninth one generated from 1991-1994 data.

---

<sup>6</sup> These were originally called Leximappes [Turner 88].



**Figure 1: First Example of a Pass-1 Network**



**Figure 2: Second Example of a Pass-1 Network**

### 3.2.2 Pass-2

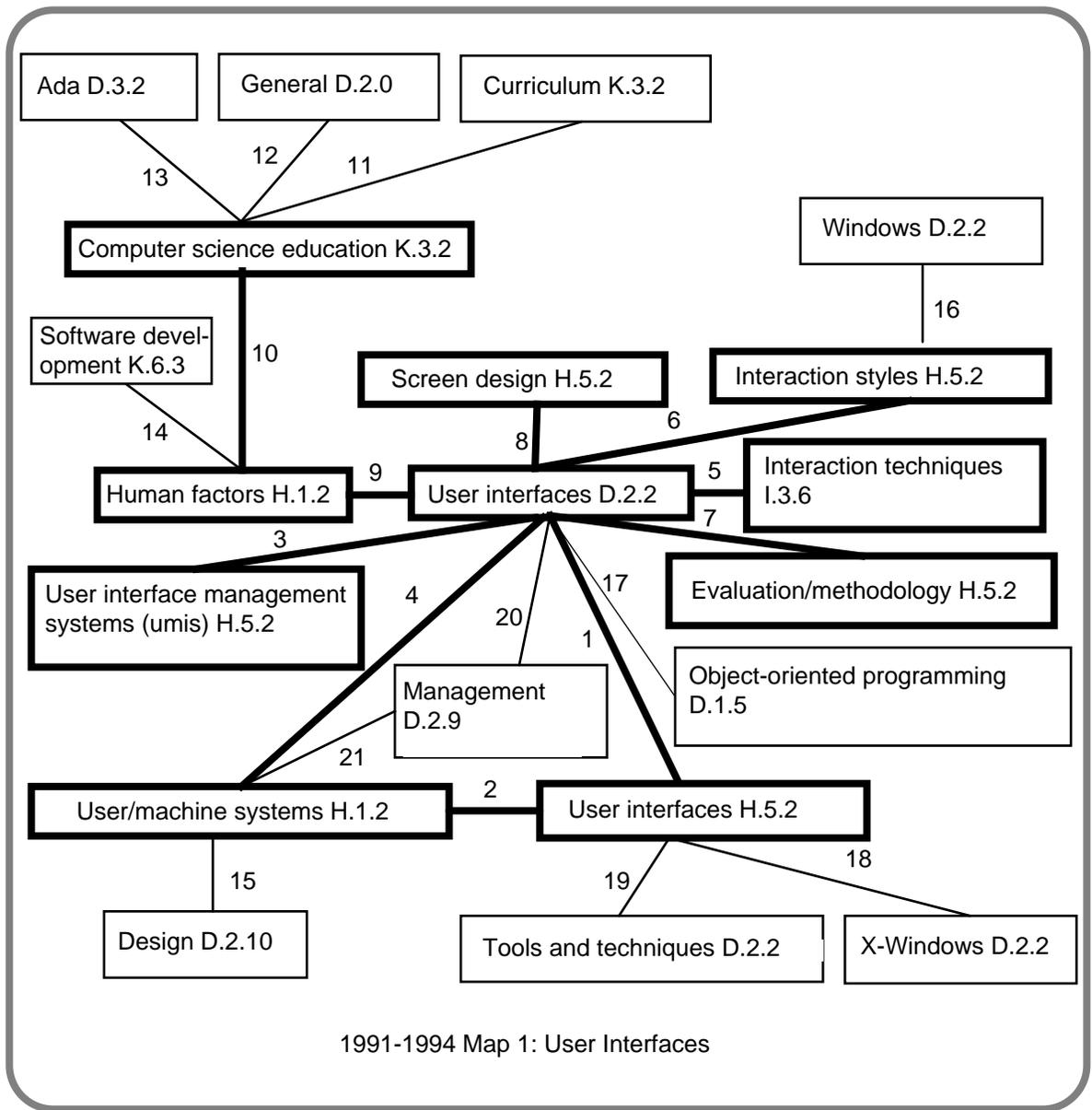
The second pass (Pass-2) is designed to seek further associations among descriptors found in Pass-1. During Pass-2, networks are extended by the addition of Pass-2 links. To be a candidate for inclusion in Pass-2, both nodes (descriptors) of a Pass-2 link must be in some Pass-1 networks. A Pass-2 link connects a Pass-1 node in a given network to a node that had occurred as a Pass-1 node in another network but is represented in the given network as a Pass-2 node.<sup>7</sup> Pass-2 nodes and Pass-2 links are represented by thin boxes and by thin lines connecting them with Pass-1 nodes, respectively. Pass-2 becomes the basis for determining how networks fit together in larger super networks (see Chapter 6, Super Network Analysis).

As in Pass-1, candidate links are included in Pass-2 based on their strengths and co-occurrence counts. The order of Pass-2 links is by descending values for qualifying links. A node can appear in only one Pass-1 network, but can appear in more than one Pass-2 link.

Figure 3 illustrates this process for Pass-2 of the network in Figure 1. Recall that Pass-2 nodes must always appear previously as Pass-1 nodes in other networks. In Figure 2, Curriculum (K.3.2) forms a Pass-2 connection with the Pass-1 node Computer Science Education (K.3.2) via link 11 in Figure 3.

---

7. Sometimes two Pass-1 nodes in a network are joined during Pass-2; such links are considered Pass-1 links because they join two Pass-1 nodes.



**Figure 3: Pass-1 and Pass-2 Nodes and Links**

Table 3 shows data for Pass-1 and Pass-2 links in Figure 3. The Pass-1 networks of all nodes incorporated during Pass-2 are given in the last column (it is 1 for all Pass-1 links). Two nodes from Figure 2 (Map 9 of 1991-1994) are in Pass-2 links of the Figure 1 network. Other links come from the various Pass-1 networks for the data.

**Table 3: Links in Decreasing Order of Strength**

Order	Node 1	Node 2	Co-Occurrence	Strength (S)	Pass-1 Map
<b>Pass -1</b>					
1	User interfaces D.2.2	User interfaces H.5.2	177	0.181802	1
2	User interfaces H.5.2	User/machine systems H.1.2	56	0.062695	1
3	User interface management systems (uims) H.5.2	User interfaces D.2.2	47	0.057496	1
4	User interfaces D.2.2	User/machine systems H.1.2	69	0.051381	1
5	Interaction techniques I.3.6	User interfaces D.2.2	32	0.036248	1
10	Computer science education K.3.2	Human factors H.1.2	20	0.029121	1
6	Interaction styles H.5.2	User interfaces D.2.2	44	0.025195	1
7	Evaluation/methodology H.5.2	User interfaces D.2.2	16	0.012586	1
8	Screen design H.5.2	User interfaces D.2.2	16	0.012246	1
9	Human factors H.1.2	User interfaces D.2.2	27	0.009487	1
<b>Pass-2</b>					
11	Computer science education K.3.2	Curriculum K.3.2	18	0.080198	9
12	Computer science education K.3.2	General D.2.0	38	0.034534	9
13	Ada D.3.2	Computer science education K.3.2	17	0.009506	7
14	Human factors H.1.2	Software development K.6.3	24	0.009167	3
15	Design D.2.10	User/machine systems H.1.2	18	0.007778	8
16	Interaction styles H.5.2	Windows D.2.2	29	0.007569	5
17	Object-oriented programming D.1.5	User interfaces D.2.2	55	0.007446	3
18	User interfaces H.5.2	X-Windows D.2.2	19	0.005405	6
19	Tools and techniques D2.2	User interfaces H.5.2	34	0.005361	7
20	Management D.2.9	User interfaces D2.2	29	0.004623	3
21	Management D.2.9	User/machine systems H.1.2	15	0.004261	3

### 3.2.3 Algorithm Constraints

Without some minimum constraints, descriptors appearing infrequently but almost always together could dominate networks; hence a minimum co-occurrence  $c_{ij}$  value is required to generate a link. At the same time, some maps can become cluttered due to an excessive number of legitimate links (but of generally decreasing S values); hence, restrictions on numbers of nodes and links are sometimes required to facilitate the discovery of major partitions of concepts. However, many networks are limited only by the number of qualifying nodes, as we will observe later.

For the time periods of 1987-1990 and 1991-1994, 15 co-occurrences of descriptors were required before they could become candidates for linking; for 1982-1986, the co-occurrence cut-off was set at 5 to accommodate the lesser volume of data. For all time periods the number of links and nodes in each network, both Pass-1 and Pass-2, was set at 24 links and 20 nodes. For these values, the co-word algorithm generated 15, 16, and 11 networks, respectively, for the periods 1982-1986, 1987-1990, and 1991-1994. Table 4 summarizes these values.

**Table 4: Parameters and Resulting Networks**

Time Period	Minimum Co-Occurrence	Maximum Nodes	Maximum Links	Networks Generated
1982-1986	5	20	24	15
1987-1990	15	20	24	16
1991-1994	15	20	24	11

### 3.2.4 Algorithm Summary

Following is a summary of the algorithm:

- 1 Select a minimum for the number of co-occurrences,  $c_{ij}$ , for descriptors  $i$  and  $j$ .
- 2 Select maxima for the number of Pass-1 links and nodes.
- 3 Select maxima for the total (Pass-1 and Pass-2) links and nodes.
- 4 Start Pass-1.
- 5 Generate the highest S value from all possible descriptors to begin a Pass-1 network.
- 6 From that link, form other links in a breadth-first manner until no more links are possible due to the co-occurrence minima or to Pass-1 link or node maxima. Remove all incorporated descriptors from the list of subsequent available Pass-1 descriptors.
- 7 Repeat Steps 5 and 6 until all Pass-1 networks are formed; i.e., until no two remaining descriptors co-occur frequently enough to begin a network.
- 8 Begin Pass-2.
- 9 Restore all Pass-1 descriptors to the list of available descriptors.
- 10 Starting with the first Pass-1 network, generate all links to Pass-1 nodes in that network with any Pass-1 nodes having at least the minimal co-occurrences in descending order of S

value; stop when no remaining descriptors meet co-occurrence minima or when total node or link maxima are met. Do not remove any descriptors from the available list.

11 Repeat Step10 for each succeeding Pass-1 network.

A maximum number of Pass-1 networks can be specified in cases where an excessive number of networks will be generated otherwise; this restriction was not necessary here.

Numerous variations of this algorithm are possible.

### **3.3 Comments on Selection of Network Parameters**

Link and node limitations mostly determine how networks will be generated in concert with the corresponding co-occurrence minimum. If the co-occurrence minimum is too high, few links may be formed; if it is too low, an excessive number of links may result. In the former case, subspecialities in a field may not emerge; in the latter case, a field may look disproportionately cluttered.

The parameters for 1982-1986 were chosen somewhat arbitrarily because of the small amount of data. We attempted to establish a baseline for comparison with following generations. The primary point of contention was the co-occurrence value of 5. It is somewhat higher in proportion to the number of documents and descriptors than the value of 15 for succeeding generations. We feel the number of networks and super networks generated supports our choice.

In setting co-occurrence values for the 1987-1990 and 1991-1994 generations, the proper values could be determined at least two ways: as a function of the ratios of indexed items or the ratio of the number of descriptors. We used the former. Because the numbers of items for the generations were almost equal (7,650 and 7,395), we set the co-occurrences the same. However, the numbers of descriptors were sufficiently different (28,471 and 23,611) to question if the co-occurrence for 1991-1994 should be lower than for 1987-1990. To test this hypothesis, we set the 1991-1994 co-occurrence at 13 and recomputed.

This change still resulted in 11 networks. Some networks were different, but only on the fringes. The central themes remained the same. More links and nodes were realized with the lower co-occurrence value (16% and 19%, respectively), as would be expected. Many of these new links and nodes were formed through additional connections of already existing nodes in the same and in other maps existing at the higher co-occurrence level. Additionally, 1 isolated network with only 2 nodes was absorbed by a larger network at the 13 co-occurrence level, while a new, isolated network with 3 nodes and 2 links emerged.

So, while the link, node, and co-occurrence parameters effectively control the generation of networks, small changes in their values appear to affect only marginal links, at least in this study. Of course, additional and subsequent data can affect the generation of core themes without changes in parameters, which is the intent of co-word analysis.

## 4 Network Analysis

Network analysis reveals the primary patterns of emphasis in a period. Representing the interactions of nodes and links within larger contexts enables the identification of research and commentary trends in software engineering publications.

Maps of all networks appear in the appendix. A wealth of information emerges from these maps, but much of it is difficult to distill. We will use a combination of qualitative and quantitative approaches in analyzing the networks.

### 4.1 Network Names

#### 4.1.1 Methodology

We named the maps in an attempt to summarize their main thrusts. This is not a precise activity. In some cases the maps had highly connected descriptors; in others, two (and sometimes three) descriptors exhibited high connectivity. In the latter cases, we used hyphenated names. We always used descriptors contained in nodes. Generally, we used descriptor(s) from the node(s) with the most connections, giving greater weight to Pass-1 nodes in close calls. We used a D.2 descriptor as a name if possible. To assist further in interpreting networks' names, other prominent descriptors are included below the primary one(s) if needed.

#### 4.1.2 Findings

The names chosen are as follows:

##### 4.1.2.1 1982-1986 Networks

1. Software Management - Ada  
Management
2. Logic Programming
3. User Interfaces  
Human factors, software psychology
4. Standards
5. Tools and Techniques - Structured Programming - Pascal
6. Software Development  
Programming environments
7. Software Libraries
8. Testing and Debugging - Correctness Proofs  
Software quality assurance, concurrent programming
9. Reliability
10. Program Editors

11. Requirements/Specifications - Systems analysis and design
12. Modules and Interfaces
13. Real-Time Systems
14. Abstract Data Types
15. Metrics
  - Life cycle

#### **4.1.2.2 1987-1990 Networks**

1. Geometrical Problems and Computations
2. Correctness Proofs - Languages
  - Semantics, real-time and embedded systems
3. Logic Programming
4. Requirements/Specifications - Methodologies
  - Program verification, abstract data types
5. User/Machine Systems
  - User interfaces, human factors
6. Methodologies - Software Development
  - Computer science education
7. Standards
8. Structured Programming
9. Applications and Expert Systems - Tools and Techniques
  - Interactive
10. Concurrent Programming - Ada
  - Compilers
11. Computer-Aided Design
12. Error Handling and Recovery
13. Distribution and Maintenance
14. Software Configuration Management
15. Reusable Software
16. Software Management - Design

#### **4.1.2.3 1991-1994 Networks**

1. User Interfaces
  - Computer science education
2. Petri Nets
3. Software Development - Object-Oriented Programming

4. Software Libraries - C++ - Microsoft Windows  
Object-oriented programming, C
5. Windows
6. X-Windows
7. Tools and Techniques - CASE - Systems Analysis and Design  
Ada, object-oriented programming, programming environments
8. Requirements/Specifications  
Testing and debugging, program verification
9. General  
Computer science education
10. Concurrent programming
11. Metrics

Perusing the maps of networks in the appendix reveals several variations in structure. Some maps have few nodes, some maps have many nodes, and some are dominated by connections from one or two nodes. Others have distributed connections; while still others are not really one map, but two (or three) maps. We will describe these variations more fully in the following section.

For reference in the following sections, the primary network names for each time period are given in Table 5. Note that networks are numbered sequentially in the order generated by co-word analysis algorithms; hence, the same numbers do not imply the same network names across time periods.

**Table 5: Network Names and Numbers**

	<b>1982-1986</b>	<b>1987-1990</b>	<b>1991-1994</b>
1	Software Management - Ada	Geometrical Problems and Computations	User Interfaces
2	Logic Programming	Correctness Proofs - Languages	Petri Nets
3	User Interfaces	Logic Programming	Software Development - Object-Oriented Programming
4	Standards	Requirements/Specifications - Methodologies	Software Libraries - C++ - Microsoft Windows
5	Tools and Techniques - Structured Programming - Pascal	User/Machine Systems	Windows
6	Software Development	Methodologies - Software Development	X-Windows
7	Software Libraries	Standards	Tools and Techniques - CASE - Systems Analysis and Design
8	Testing and Debugging - Correctness Proofs	Structured Programming	Requirements/Specifications
9	Reliability	Applications and Expert Systems - Tools and Techniques	General
10	Program Editors	Concurrent Programming - Ada	Concurrent Programming
11	Requirements/Specifications - Systems Analysis and Design	Computer-Aided Design	Metrics
12	Modules and Interfaces	Error Handling and Recovery	
13	Real-Time Systems	Distribution and Maintenance	
14	Abstract Data Types	Software Configuration Management	
15	Metrics	Reusable Software	
16		Software Management - Design	

## 4.2 Network Summaries

### 4.2.1 Methodology

When applied to the CCS data, the co-word analysis algorithm produced networks of varying complexities in terms of numbers of nodes and links, and numbers of documents per network. From these basic measures, we can examine properties of networks and their corresponding maps.

First, consider the complexity of the networks in terms of nodes and links. The ratio of links to nodes  $L/N$  is a measure of the complexity of a network. Note that  $(N-1)/N \leq L/N \leq (N-1)/2$ , where  $N \geq 2, L \geq 1$ . Hence, the minimum value for  $L/N$  is  $1/2$ . We observe that the ratios of links to nodes generally increases as the number of nodes increases, but remains less than 2 even for the larger networks; many potential links are unrealized.

A related, normalized measure of a network's complexity is its "percentage of connectivity." This metric is based on the ratio of the number of links in a network to its maximum possible number of links,  $2L/(N(N-1))$ . It is greater for simple stand-alone networks or for subnetworks of larger networks because of the fewer numbers of nodes and links.

The maps show only connections; they do not show how many documents in the corpus are covered by the networks. The tables show this feature for individual networks and for all networks in a time period. These measures are important because they reveal much about the degree of interactions of documents and descriptors in the corpus.

### 4.2.2 Findings

The results of this analysis are presented in Tables 6, 7, and 8.

Consider the 1982-1986 data in Table 6. Here, 698 unique documents were covered by the 15 networks, from a total of 1,646 documents (42%); this means 698 documents had co-occurring descriptors that appeared in at least 4 other documents for this time period.

Now consider Map 1 in Table 6: it covers 136 unique documents (19% of the 698 total used). Notice that the networks with higher  $L/N$  ratios have the higher document and node values. Also note that the column "Percentage of Documents" totals to more than 100% because a document can be included in the construction of more than one map.

**Table 6: 1982-1986 Network Summary Data**

Total unique documents included: 698						
Total documents available:1646						
Percentage of documents used:42%						
Map	Nodes <i>N</i>	Links <i>L</i>	<i>L/N</i>	Percentage of Connectivity	Unique Documents	Percentage of Documents <sup>a</sup>
1	18	24	1.33	16%	136	19%
2	2	1	0.50	100%	6	1%
3	17	24	1.41	18%	197	28%
4	2	1	0.50	100%	5	1%
5	20	23	1.15	12%	108	15%
6	20	22	1.10	12%	173	25%
7	3	2	0.67	67%	11	2%
8	20	23	1.15	12%	129	18%
9	6	6	1.00	40%	16	2%
10	3	2	0.67	67%	13	2%
11	17	24	1.41	18%	117	17%
12	2	1	0.50	100%	5	1%
13	3	2	0.67	67%	12	2%
14	3	2	0.67	67%	9	1%
15	12	13	1.08	20%	55	8%
				Totals	992	142% <sup>a</sup>

a. Can exceed 100% because a document can be included in more than one network.

**Table 7: 1987-1990 Network Summary Data**

Total unique documents included: 3062						
Total documents available:7650						
Percentage of documents used:40%						
Map	Nodes <i>N</i>	Links <i>L</i>	<i>L/N</i>	Percentage of Connectivity	Unique Documents	Percentage of Documents <sup>a</sup>
1	6	6	1.00	40%	50	2%
2	17	22	1.29	16%	251	8%
3	3	2	0.67	67%	27	1%
4	16	24	1.50	20%	485	16%
5	15	24	1.60	23%	847	28%
6	20	23	1.15	12%	732	24%
7	3	2	0.67	67%	29	1%
8	4	3	0.75	50%	67	2%
9	19	24	1.26	14%	666	22%
10	18	24	1.33	16%	396	13%
11	4	3	0.75	50%	52	2%
12	2	1	0.50	100%	17	1%
13	4	3	0.75	50%	52	2%
14	6	5	0.83	33%	75	2%
15	16	24	1.50	20%	422	14%
16	11	20	1.82	36%	324	11%
				Totals	4492	147% <sup>a</sup>

a. Can exceed 100% because a document can be included in more than one network.

**Table 8: 1991-1994 Network Summary Data**

Total unique documents included:2881						
Total documents available:7395						
% documents used:38%						
Map	Nodes <i>N</i>	Links <i>L</i>	<i>L/N</i>	Percentage of Connectivity	Unique Documents	Percentage of Documents <sup>a</sup>
1	20	21	1.05	11%	565	20%
2	2	1	0.50	100%	27	1%
3	20	23	1.15	12%	861	31%
4	17	24	1.41	18%	492	18%
5	17	17	1.00	13%	401	14%
6	8	7	0.88	25%	125	4%
7	17	24	1.41	18%	643	23%
8	16	23	1.44	19%	487	17%
9	5	5	1.00	50%	95	3%
10	4	3	0.75	50%	37	1%
11	5	5	1.00	50%	83	3%
				Totals	3816	136% <sup>a</sup>

a. Can exceed 100% because a document can be included in more than one network.

These data show the variation in network structures within a time period. Some networks are minimal; they have only two nodes. Examples are Network-2, -12, and -2 (Logic Programming, Error Handling and Recovery, and Petri Nets) from 1982-1986, 1987-1990, and 1991-1994, respectively. Some other networks approach minimal structure.

Other networks are more fully formed. Some embody the maximum allowable number of links, nodes, or both. See 1991-1994 Network-1,-3, -4, -7, and -8 (User Interfaces, Software Development - Object Oriented Programming, Software Libraries - C++ - Microsoft Windows, Tools and Techniques - CASE - Systems Analysis and Design, Requirements/Specification) for examples.

## 5 Types of Networks and Their Interactions

### 5.1 Methodology

There are essentially three types of networks: principal, secondary, and isolated. Principal networks are connected to one or more (secondary) networks. Secondary networks generally are linked to principal networks through a relatively high number of external links in the principal networks. Isolated networks have an absence (or low intensity) of links with other networks.

Isolated networks often have links with high  $S$  values, usually accompanied by low co-occurrence  $c_{ij}$  values. While isolated networks are easy to recognize, principal and secondary networks may not be. Therefore, we will define and operationalize terms that characterize these functionalities.

We defined *density* as the mean of the Pass-1  $S$  values of a network; *centrality* is defined as the square root of the sum of the squares of the Pass-2  $S$  values of a network in order to distinguish among relatively close values. Density represents the internal strength of a network, while centrality represents a network's position in strength of interaction with other networks.<sup>8</sup>

### 5.2 Findings

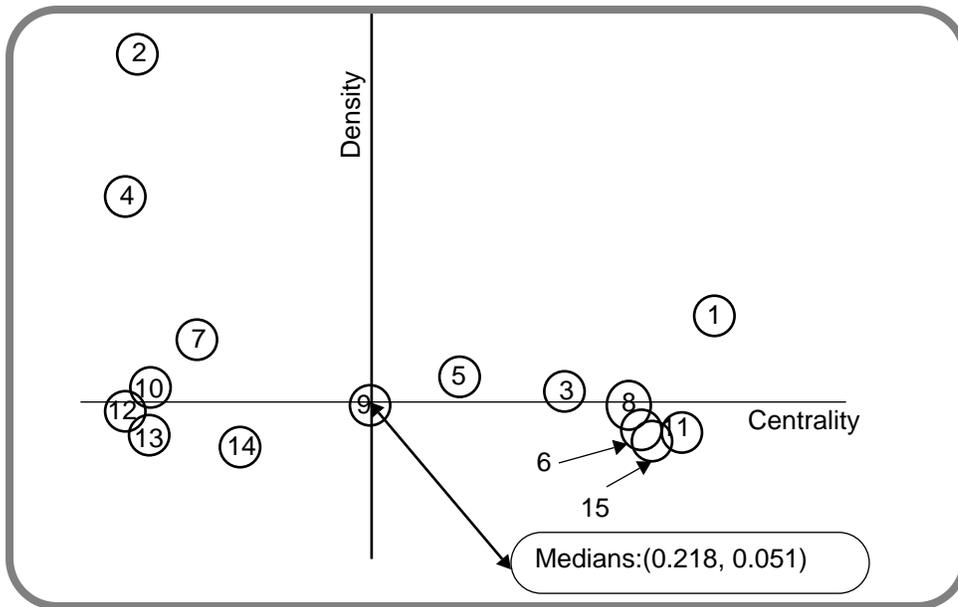
Plots of centrality and density for each of the time periods are shown in Figures 4, 5, and 6.<sup>9</sup> The origin of these figures is the median of the respective axis values (the horizontal axis represents centrality; the vertical axis represents density). Not surprisingly, most networks with strong centrality scores also show relatively high unique document counts and  $L/N$  ratios, as indicated in Table 5 for 1991-1994 data.

Isolated networks show relatively low document counts and  $L/N$  ratios. (see 1991-1994 Network-2, Petri Nets).

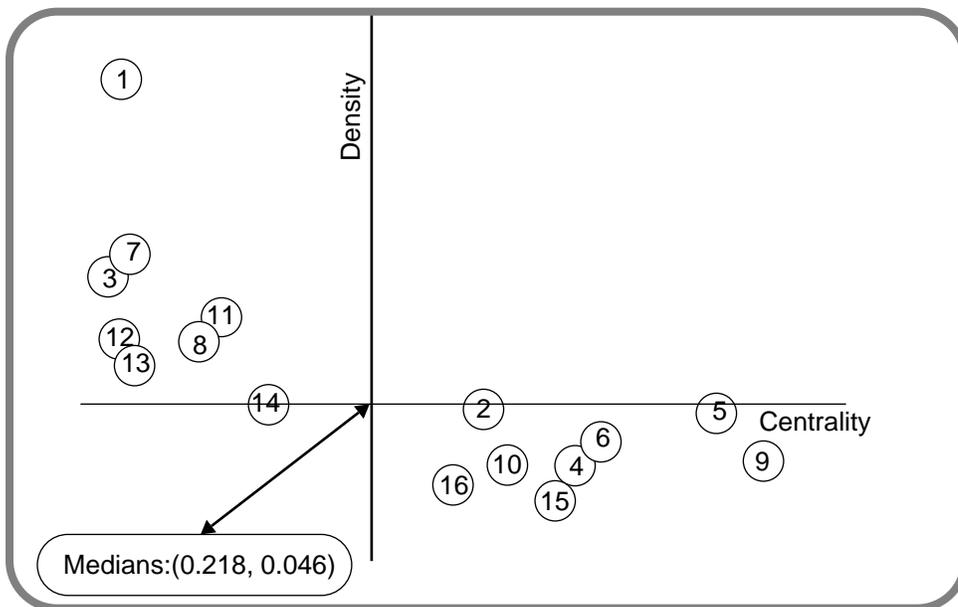
---

8. These terms are accepted ones in co-word analysis literature. We recognize that density and centrality have others domain-specific connotations--say, in statistics. Alternative choices include *density* and *centrality*, but these already have meanings in software engineering literature. *Adhesion* and *density* could be used, but that is not conventional in co-word literature, and it confounds the use of the term density even more. We trust the intent of the terminology is clear.

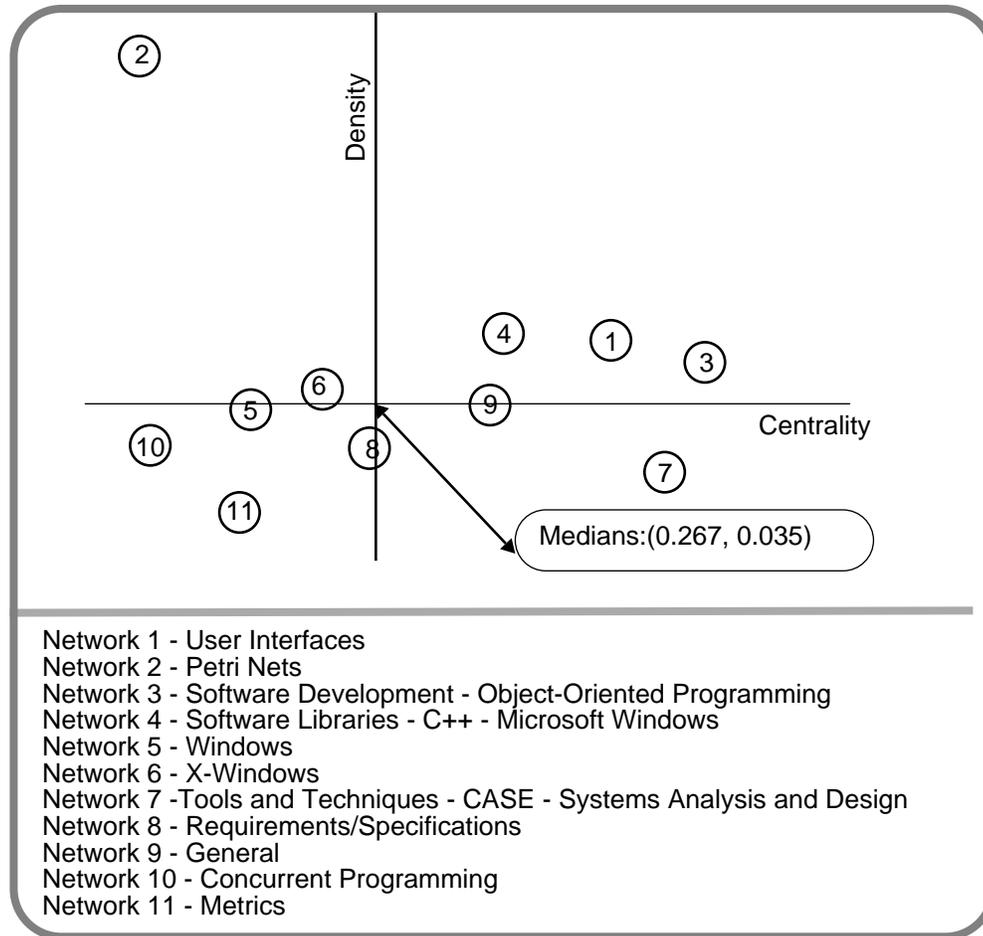
9. Figures 4, 5, and 6 are not to precise scale; relative positions are represented.



**Figure 4: 1982-1986 Centrality and Density**



**Figure 5: 1987-1990 Centrality and Density**



**Figure 6: 1991-1994 Centrality and Density**

Perhaps the most interesting networks are the ones with both strong density and strong centrality. Few of these emerge, which testifies to software engineering's somewhat indefinite focus. None are identified in 1987-1990. However, in the 1991-1994 data, Networks-1, -3, and -4 have these properties. These networks also have strong interaction with each other. Network-7 shows strong centrality. Network-2 shows strong density but weak (actually, zero) centrality values; note that it is a completely isolated network, so its position matches intuition. Network-10 and Network-11 are below the median for both centrality and density scores. Similar analyses can be performed on the other periods.

### 5.3 Evidence of a Coalescing Field

Indications are that software engineering is finding more general definition in 1991-1994 than in the other two earlier time periods. We can see this by looking at data for numbers of net-

works, for centrality, and for density in Table 9.

**Table 9: Comparison of Properties for Time Periods**

Property	1982-1986	1987-1990	1991-1994
Number of Networks	15	16	11
Median Centrality	.2176	.2176	.2664
Median Density	.0507	.0458	.0350

This comparison is especially striking for the periods 1987-1990 and 1991-1994. We observe that the number of networks declined, the centrality measure increased, and the density measure decreased. This indicates more integration of subtopics and fewer isolated networks, as would be expected in a more focused discipline. Future data will be needed to evaluate this possible trend.

## 6 Super Network Analysis

### 6.1 Methodology

In addition to describing how networks compare within a period, we can be more specific in describing how networks interact with other specific networks; this addresses centrality in a more focused fashion, but does not substitute for the general centrality measure.

We chose to operationalize principal and secondary networks as follows: If Network-A has internal nodes that are Pass-2 nodes in  $x$  links of Network-B, and each of these links has a Pass-2  $S$  value that exceeds the minimum Pass-1  $S$  value of Network-B, then Network-A is a secondary network of Network-B.

Using this way of determining principal and secondary networks, we can describe super networks of networks. The relationships in these super networks are not inherently bi-directional, as are network links (at least as defined using  $S$ ).

### 6.2 Findings

Tables 10, 11, and 12 give all networks that have at least one qualifying connection with other networks. Shown with each network is an entry in the form  $y(z)$ ;  $y$  indicates the associated network and  $z$  shows the number of qualifying links. From this, we can then construct a super network at whatever *threshold* of  $x$  we choose.

Setting the threshold at  $x = 2$  qualifying connections, we can construct a super network of networks for each period as shown in Figures 7, 8, and 9. By selecting higher or lower values for the threshold (either in terms of the number of qualifying links or the level of qualification), we can derive other super networks.

Consider the 1991-1994 super network (Figure 9) and its underlying generating data (Table 12). The names and other prominent descriptors of 1991-1994 networks are included in Figure 9 for convenience because they are used in the following discussion.

Some observations include the following:

- Network-2, -5, -6, -10, and -11 are isolated networks.
- Network-3 is a secondary network of principal network Network-8; Network-7 is a secondary network of Network-8.
- Network-3 is a principal network and a secondary network relative to both Network-4 and Network-7.
- Network-7 is especially strongly connected to Network-3; Network-3 is less strongly connected to Network-7 (at least relative to the former).

Putting this in context of the networks' contents, we might conclude the following:

- Object-oriented programming is a major focus of software development.
- Software libraries have combined with object-oriented methodologies as principal development activities.
- The major systems used now in software engineering are Ada, C++, C, UNIX, X-windows, and Microsoft Windows.
- Computer-aided software engineering and object-oriented languages are emerging as specific tools in software development.

Looking further at the isolated networks and the centrality/density diagram, we might conclude that Petri Nets is either an emerging or dying research topic because it is completely isolated from other networks. Many other conclusions and impressions are derivable from the networks and super networks. Interested readers can make additional analyses with the information provided.

**Table 10: Possible 1982-1986 Super Networks**

<b>Possible 1982-1986 Super Networks</b>	
<b>Network</b>	<b>Connected Networks [network number(number of links)]</b>
1	3(1), 6(2), 11(2), 15(1)
2	none
3	1(1), 15(2)
4	none
5	6(2)
6	1(2), 5(2), 7(1), 8(1), 10(1), 11(2), 15(1)
7	none
8	11(1), 14(1)
9	none
10	none
11	1(2), 6(2), 8(2)
12	none
13	none
14	8(1)
15	1(1), 3(2), 8(1)

**Table 11: Possible 1991-1994 Super Networks**

<b>Possible 1991-1994 Super Networks</b>	
<b>Network</b>	<b>Connected Networks [network number(number of links)]</b>
1	7(1), 9(2)
2	none
3	4(2), 7(5)
4	3(3), 6(1)
5	none
6	4(1)
7	1(1), 3(13)
8	1(1), 3(5), 7(3)
9	1(2)
10	none
11	none

**Table 12: Possible 1987-1990 Super Networks**

<b>Possible 1987-1990 Super Networks</b>	
<b>Network</b>	<b>Connected Networks [network number(number of links)]</b>
1	none
2	4(3)
3	none
4	2(4), 6(5), 15(2), 16(1)
5	9(4)
6	4(2), 5(2), 9(5), 14(1), 16(1)
7	none
8	none
9	5(7), 6(5), 10(1), 15(1)
10	2(1), 6(1), 9(1), 15(2)
11	none
12	none
13	none
14	none
15	4(1), 6(2), 9(3), 10(3)
16	6(1)

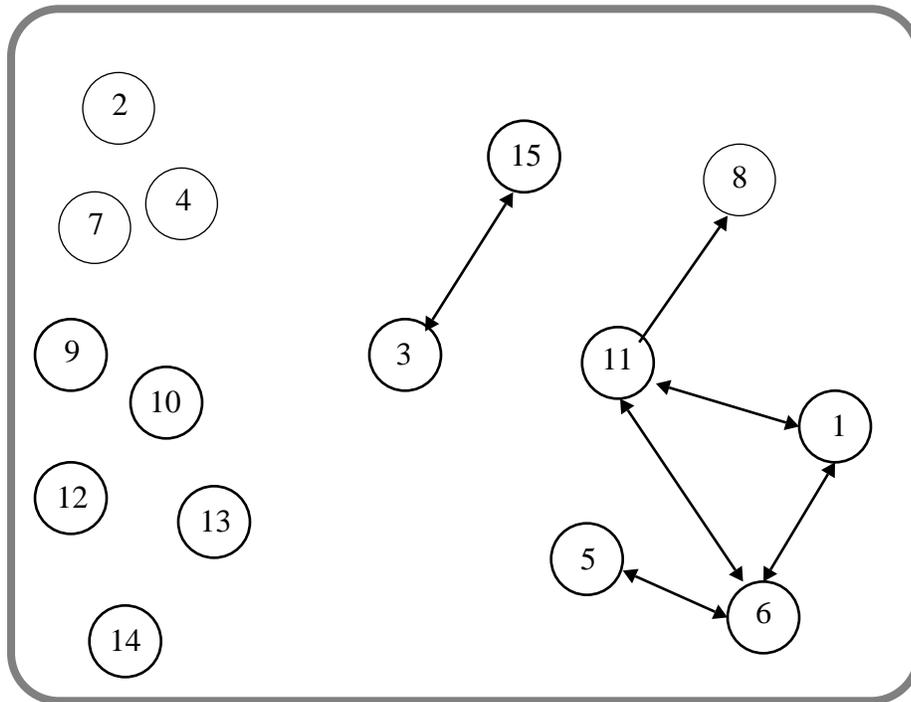


Figure 7: 1982-1986 Super Network,  $x = 2$

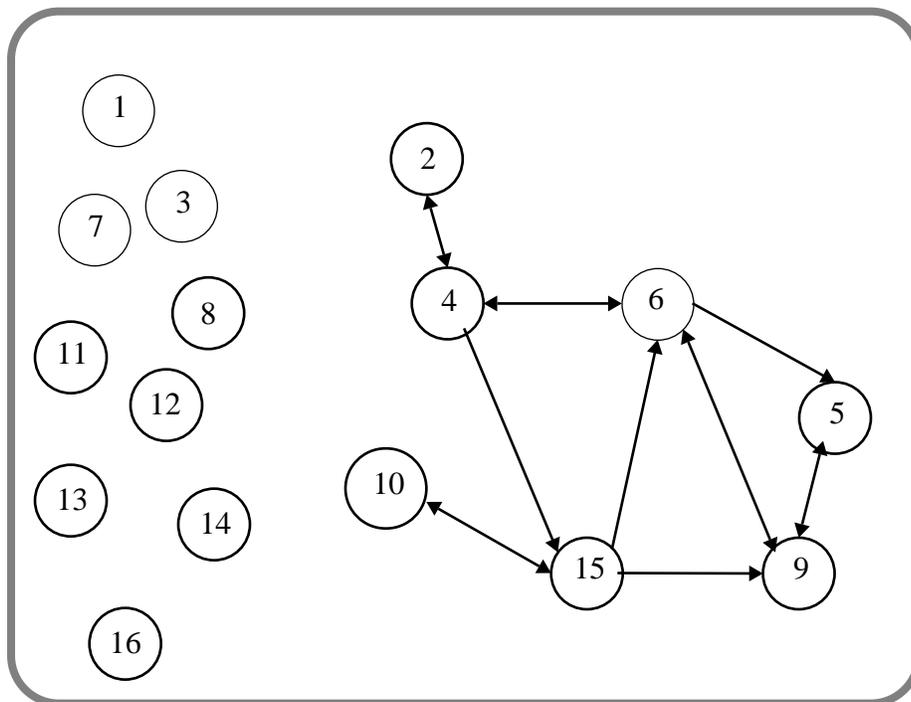
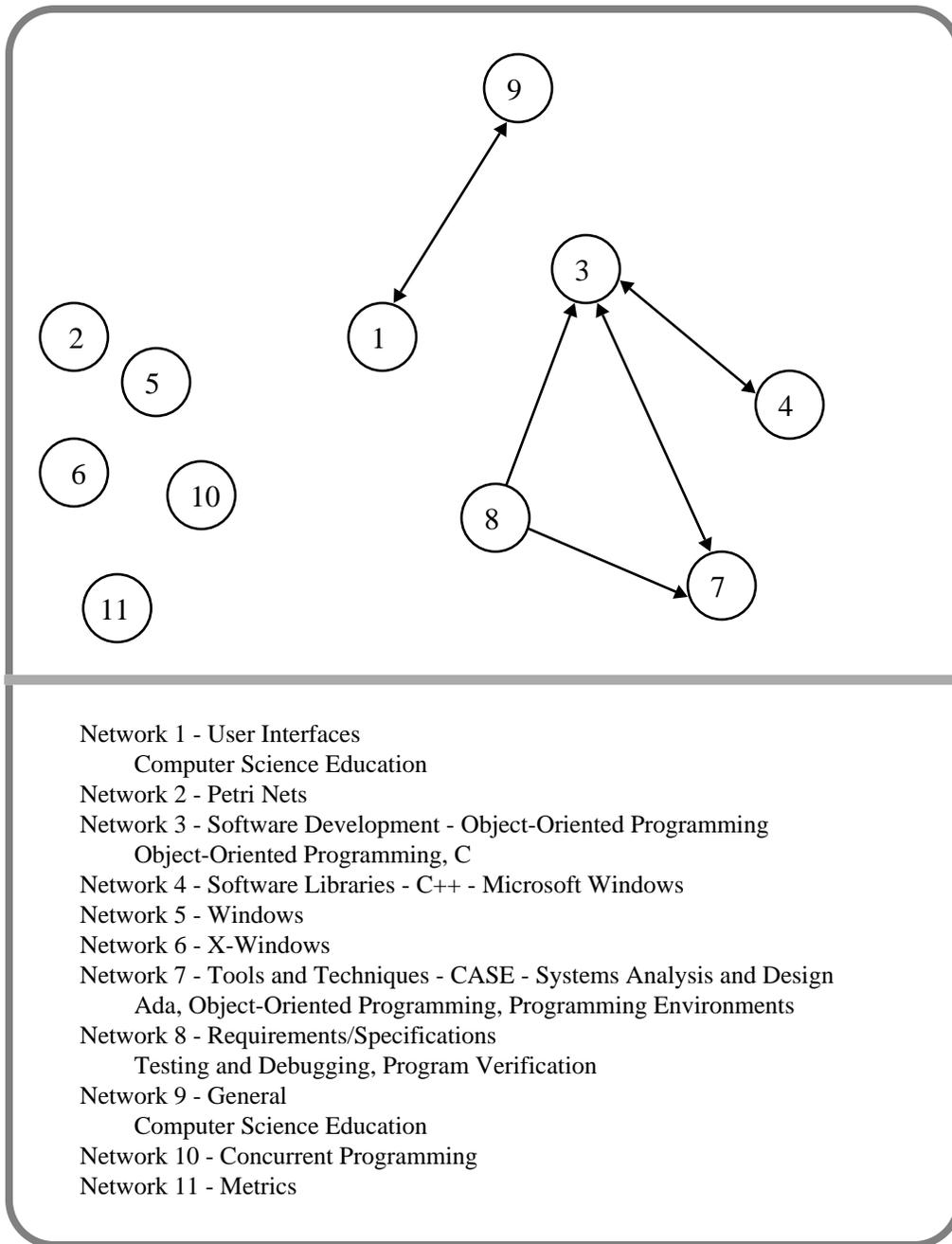


Figure 8: 1987-1990 Super Network,  $x = 2$



**Figure 9: 1991-1994 Super Network,  $x = 2$**

## 7 Trends over Periods

By examining the super networks and their component networks over the different time periods, we can observe aspects of the evolution of software engineering. First we consider specific contexts of some descriptors in different time periods; then we illustrate a way to trace the transformation of general network themes over time.

This general methodology is applicable in other similar applications. We demonstrate this technique for CCS findings.

### 7.1 Analysis of Descriptor Contexts

Through the use of network names, we observed that the foci of study in each period were software development (which includes management), user interfaces, parallelism, verification and validation, requirements/specifications, and tools and techniques. However, while these foci maintain some of the same connections over different time periods, they also evolve by forming new connections to different nodes. For example, the 1991-1994 Network-7 (Tools and Techniques) appears with CASE, objected-oriented techniques, reuse, and Ada; whereas in the related 1982-1986 Network 5, Tools and Techniques appears with Pascal and structured programming topics.

Much of the change can be gleaned from detailed examinations of networks. To illustrate this process, we will present two detailed cases.

First, we will look at some smaller portions of pertinent networks. In 1982-86, Ada appears as four nodes in Network-1 (Software Management - Ada)<sup>10</sup> but in a rather isolated fashion (Figure 10). Later, it becomes an integral part of 1987-1990 Network-15 (Reusable Software) (Figure 11) and 1991-1994 Network-7 (Tools and Techniques - CASE - Systems Analysis and Design) (Figure 12); in the middle time period, it associates with high-level concepts such as software reuse, software libraries, module interfaces, concurrency, and object-oriented software. In the latter period, it associates with military, which demonstrates Ada's special importance in that arena of software development, and with computer science education, which attests to Ada's general importance in the research community; its association with reusable software continues.<sup>11</sup>

---

<sup>10.</sup> As an implicit subject descriptor, Ada can appear in any appropriate CCS category. It was the object of study from many approaches at the time.

<sup>11.</sup> Ada appears in several networks during each time period; these networks and contexts are selected as examples.

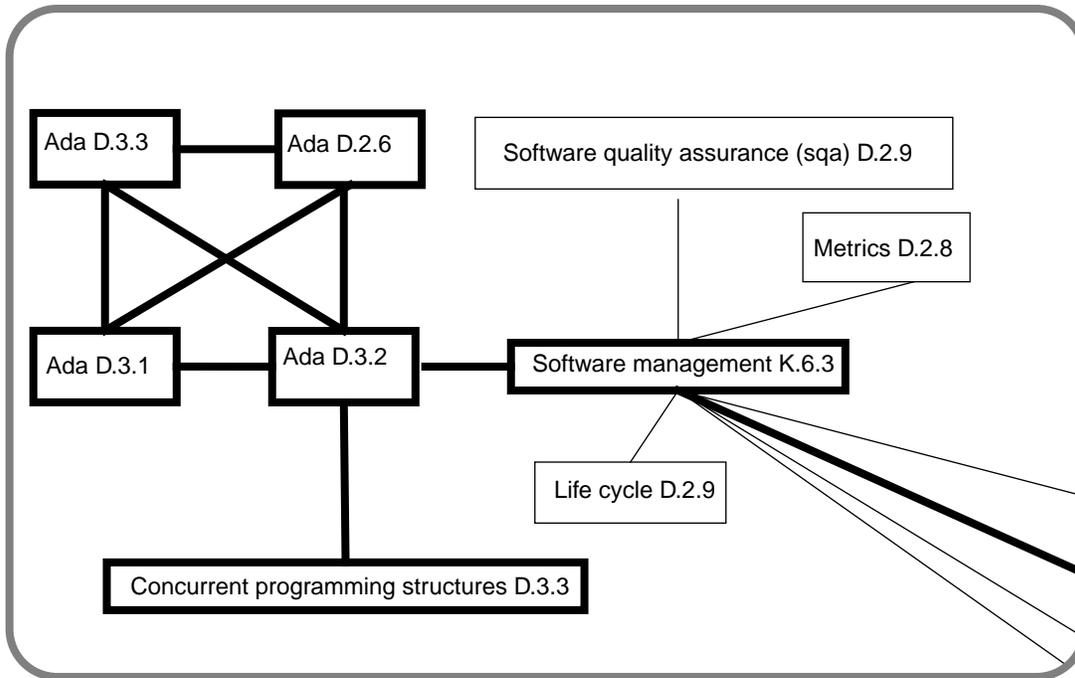


Figure 10: Ada in Network-1, 1982-1986

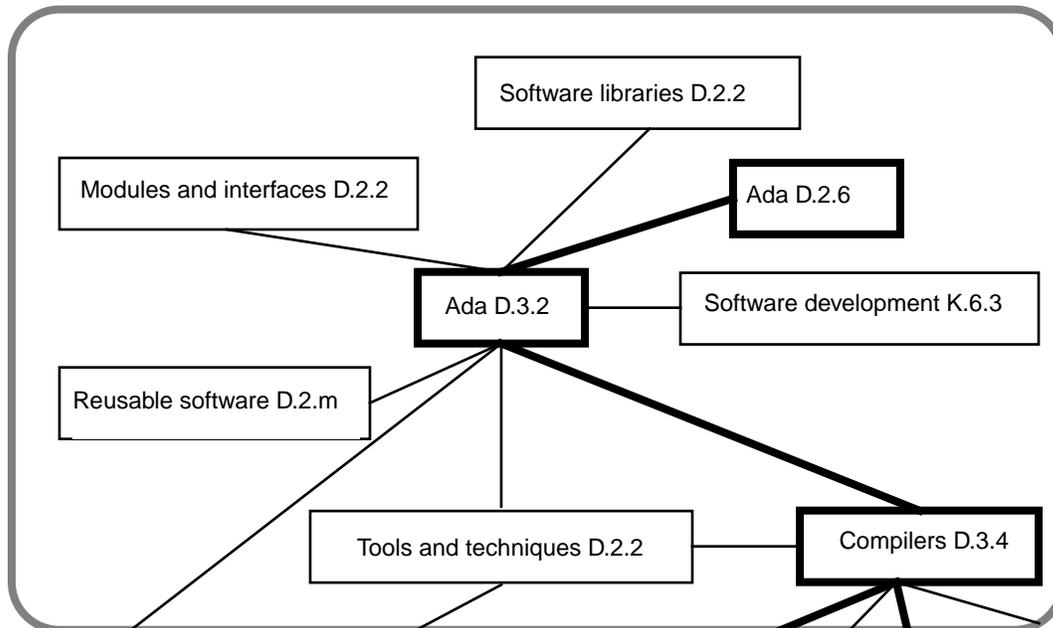
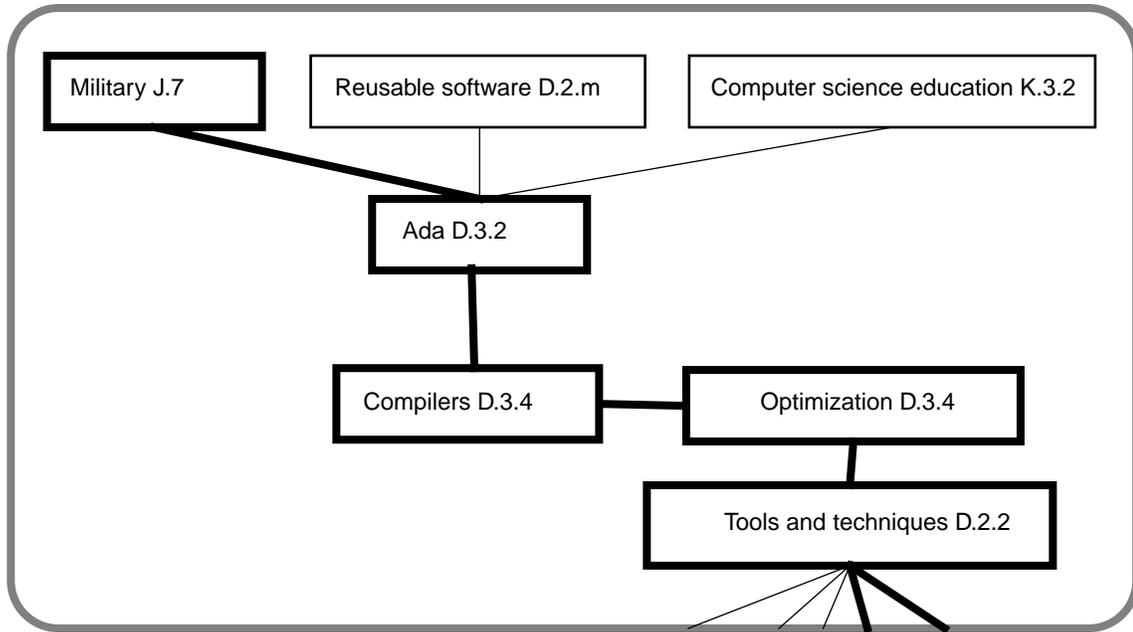
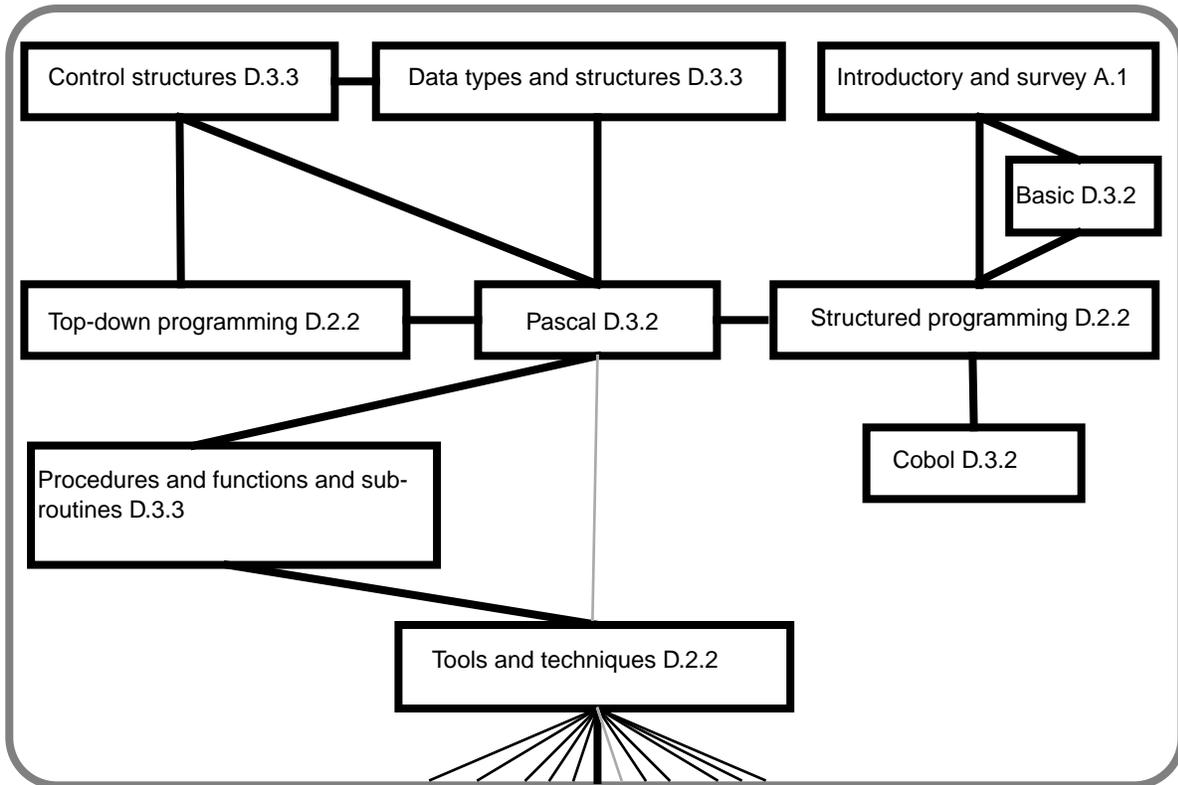


Figure 11: Ada in Network-10, 1987-1990

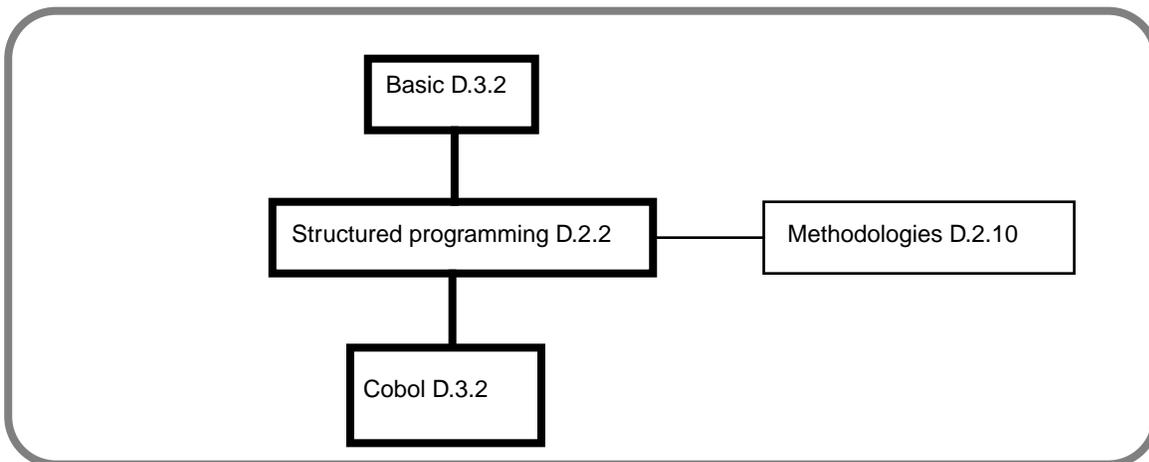
As high-level software issues become more integrated, older issues fade. Pascal, Basic, and Cobol appear in the 1982-1986 Network-5 (Tools and Techniques - Structured Programming - Pascal; see Figure 13). This network is based on programming-in-the-small issues, such as structured programming and top-down programming. In the 1987-1990 Network-8 (Structured Programming), Basic and Cobol appear almost in isolation with structured programming (see Figure 14). That theme then disappears in 1991-1994 as software engineering research moves to programming-in-the-large concerns.



**Figure 12: Ada in Network-7, 1991-1994**



**Figure 13: Structured Programming in Network-5, 1982-1986**



**Figure 14: Structured Programming in Network-8, 1987-1990**

We can summarize some other observations; the reader may reference the corresponding maps in the appendix.

The topic of standards, which is important in any well defined engineering field, appears in isolation in 1982-1986 and 1987-1990 (Network-4 and -7, respectively, both named Standards), then goes away in 1991-1994. This indicates that standards have not been integrated into other important software engineering discussions in any of the time periods and even cease to be discussed with any regularity in the most recent time period, though not all the most recent data have been analyzed.

Petri nets and unbound action devices appear in isolation in Network-2, 1991-1994. We cannot tell if they will be part of a larger network yet. Modules and interfaces appear in isolation in Network-12, 1982-1986; then appear more interrelated with other descriptors, e.g., with Ada in 1987-1990 Network-10 (Concurrent Programming - Ada) and with reusable software in 1987-1990 Network-15 (Software Management - Design). After that, modules and interfaces do not appear, but reusable software and related themes are dominating; perhaps the topic of modules and interfaces has been subsumed in these expanded topics. We will return to this in a later chapter of this report.

The networks and contexts discussed here are not exhaustive. Many other transformations of themes are suggested by the networks and their maps.

## 7.2 Analysis of Networks Across Time Periods

### 7.2.1 Methodology

The transformation of networks and their intersections with other networks across time periods provides insights into the emergence of software engineering research themes. To quantify this analysis, we apply the *similarity index* ( $SI$ ) approach, which is patterned after Callon's dissimilarity index. [Callon 91].

$SI$  measures the intersection of the descriptors in two networks. It does not directly include the corresponding links in networks; however, since all descriptors in a network are at least indirectly linked, this metric captures some portion of network similarity.

Consider two networks  $N_i$  and  $N_j$ . Let  $w_i$  be the number of descriptors in  $N_i$ , let  $w_j$  be the number of descriptors in  $N_j$ , and let  $w_{ij}$  be the number of descriptors common to  $N_i$  and  $N_j$ . Then,

$$SI(w_i, w_j, w_{ij}) = 2 \times \left( \frac{w_{ij}}{w_i + w_j} \right), \quad 0 < SI \leq 1.$$

We multiply by 2 so that the maximum value of  $SI$  is 1, which occurs when  $N_i$  and  $N_j$  have identical nodes.

### 7.2.2 Findings

We can apply  $SI$  to examine the emergence of some 1991-1994 networks. Especially interesting are the three networks showing both strong centrality and density values (called core networks or core themes). The networks are Network-1, User Interfaces; Network-3, Software

Development - Object-Oriented Programming; and Network-4, Software Libraries - C++ - Microsoft Windows.

First, consider 1991-1994 Network-1, User Interfaces. It has reportable *SI* intersections with four 1987-1990 networks, as shown below.<sup>12</sup>

$N_1$	$N_2$	$w_1$	$w_2$	$w_{12}$	<i>SI</i>
1991-1994 Network 1	1987-1990 Networks				
Requirements Specifications- Systems Analysis and Design	1. Network-5: User/Machine Systems	14	14	6	0.423
	2. Network-6: Methodologies - Software Development	14	20	7	0.412
	3. Network-9: Applications and Expert Systems - Tools and Techniques	14	19	5	0.303
	4. Network-16: Software Management	14	11	5	0.400

Hence, the 1991-1994 theme User Interfaces incorporates descriptors from several 1987-1990 networks. Its emergence history is complicated; tracing it further could require investigation of four 1987-1990 networks and of all their 1982-1986 predecessor networks.

Similarly, 1991-1994 Network-3, Software Development - Object-Oriented Programming, displays a multiply engendered network history. It has reportable *SI* values with seven 1987-1990 networks.

<sup>12.</sup> Only CCS descriptors defined in both pertinent time periods are included in *SI* descriptor counts. To ensure notable intersection between  $N_i$  and  $N_j$ , we require  $w_{ij} \geq 5$  before reporting *SI*.

$N_1$	$N_2$	$w_1$	$w_2$	$w_{12}$	$SI$
1991-1994 Network 3	1987-1990 Networks				
Software Development - Object-Oriented Programming	1. Network-4: Requirements/ Specification - Methodologies	18	16	5	0.294
	2. Network-6: Software Development	18	20	7	0.369
	3. Network-9: Applications and Expert Systems - Tools and Techniques	18	19	5	0.270
	4. Network-10: Concurrent Programming - Ada	18	18	7	0.389
	5. Network 14: Software Configuration Management	18	6	6	0.500
	6. Network 15: Reusable Software	18	16	7	0.417
	7. Network 16: Software Management - Design	18	11	8	0.552

Observe that 1987-1990 Network-14, Software Configuration Management, was completely absorbed by the 1991-1994 network (i.e., all descriptors of the earlier network are descriptors of the latter network).

Now, consider 1991-1994 Network-4 Software Libraries - C++ - Microsoft Windows. It has a reportable  $SI$  value, 0.375, for only one 1987-1990 network, Network-15, Reusable Software. Tracing this latter network to 1982-1986 ones shows that it has a reportable  $SI$  value, 0.343, only for 1982-1986 Network-6, Software Development. This 1987-1990 network also absorbs 1982-1986 Network-7, Software Libraries, and Network-12, Modules and Interfaces; but each of these networks has fewer than five descriptors, so criteria for  $SI$  scores are not met. This history suggests a relatively well-defined emergence path for themes dealing with software reuse.

$SI$  analysis can also show the lack of a traceable past. Consider 1991-1994 Network-6, X-Windows. It has no identifiable 1987-1990 predecessors. Only four networks from that earlier period share even one descriptor with it (in all cases the same descriptor--(User interfaces d.2.2)). Similarly, 1991-1994 Network-5, Windows, has no reportable 1987-1990 predecessors. Only two 1987-1990 networks share any descriptors with it (1987-1990 Network-10 and -15, with one and two descriptors, respectively). Taken together, we see a rapid emergence of windows-based research. Sometimes research foci emerge quickly, as expected in a dynamic field.

### 7.2.3 Similarity Index Within a Time Period

*SI* can also be useful within a time period to assess the similarity of companion networks. Consider the 1991-1994 core networks: They have substantial intersection with each other, as seen below:

$N_1$	$N_2$	$w_1$	$w_2$	$w_{12}$	$SI$
1991-1994 Network 1 User Interfaces	1991-1994 Network 3 Software Development Object-Oriented Programming	20	20	7	0.350
1991-1994 Network 1 User Interfaces	1991-1994 Network 4 Software Libraries -C++ - Microsoft Windows	20	17	5	0.270
1991-1994 Network 3 Software Development Object-Oriented Programming	1991-1994 Network 4 Software Libraries - C++ - Microsoft Windows	20	17	6	0.324

The network predecessors of 1991-1994 core themes demonstrate notable characteristics. All of them with reportable *SI* scores also have high centrality scores (Figures 4, 5, and 6) for the time periods of interest, except for 1987-1990 Network-14, which had a slightly below-median score. However, that network was completely absorbed by its successor. Similarly, two 1982-1986 networks with below-median centrality scores were completely absorbed by their successor, even though their *SI* scores were not reportable. In these latter cases, the networks were all small and relatively isolated.

This observation suggests that core themes may normally emerge from predecessor networks that already display relatively strong connections to other networks within the same time period. It also suggests that isolated networks may quickly become part of more integrated networks in a succeeding time period. This absorption could occur because one new link connects a small, isolated network to a larger network. However, certainly not all isolated networks merge with larger ones, as is so far evident of the Standards networks of 1982-1986 and 1987-1990. As noted above in the case of the Structured Programming theme, a network also can transform from a core theme (1982-1986, Network-5) to an isolated theme (1987-1990, Network-8).

## 8 Descriptor Analysis

Direct analysis of co-word generated descriptor nodes gives a supporting view of which descriptors in CCS—but outside of software engineering—interact with software engineering descriptors.

### 8.1 Analysis

Recall that only descriptors that co-occur with other descriptors a requisite number of times and with relatively high strength are candidates for inclusion in networks. Many descriptors that appear in documents do not associate often enough or strongly enough with other descriptors to be considered for inclusion. The strengths of associations relative to other associations further limit which links enter into a network. Of the 1,606 unique descriptors appearing in all documents, 158 (9.8%) descriptors satisfied these criteria and appeared in the generated networks. We cannot define the maximum possible number of nodes because of unrestricted numbers of implicit subject descriptors.

Table 13 summarizes the most frequently appearing descriptors in each time period. The table was generated by first obtaining the 15 most frequently appearing descriptors within each time period and then eliminating redundancy from the combined lists. Table 13 lists descriptors alphabetically. For each descriptor, its rank in each period is shown by the number of documents in which it appears, the number of networks in which it appears, and the number of times it appears (a descriptor can be connected to more than one other descriptor in the same network, as evident in Figure 3).

**Table 13: Summary of Descriptor Data**

<b>Rank Order of Descriptor Statistics by Generation</b>									
<b>Descriptor</b>	<b>Rank in # Documents</b>			<b>Rank in # Networks</b>			<b>Rank in # times in Network</b>		
	<b>82-86</b>	<b>87-90</b>	<b>91-94</b>	<b>82-86</b>	<b>87-90</b>	<b>91-94</b>	<b>82-86</b>	<b>87-90</b>	<b>91-94</b>
Ada D.3.2	8	13	15	7t	10t	3t	13	6t	7
Applications and expert sys I.2.1	-	15	-	-	6t	-	-	5	-
Computer aided... (CASE) D.2.2	#	#	11	#	#	11t	#	#	8t
Correctness proofs D.2.4	14	-	-	11t	-	-	10t	-	-
Design D.2.1	#	14	-	#	10t	-	#	14	-
General D.2.0	3	7	7	7t	10t	11	10t	12t	15
Human factors H.1.2	6	8	-	7t	6t	-	10t	12t	-
Interactive D.2.6	-	12	-	-	10t	-	-	6t	-
Management D.2.9	9t	-	13	4t	-	5t	5t	-	8t
Methodologies D.2.10	#	3	10	#	1	11t	#	1	12t
Metrics D.2.8	15	-	-	7t	-	-	8t	-	-
Object-oriented programming D.1.5	#	#	2	#	#	2	#	#	1t
Program verification D.2.4	13	-	-	11t	-	-	14t	-	-
Programming environments D.2.6	2	4	6	1t	4	8t	5t	9t	12t
Requirements/specifications D.2.1	12	6	8	6	6t	15	4	6t	6
Reusable software D.2.m	-	9	9	-	6t	5t	-	4	8t
Software development K.6.3	4	5	5	1t	2	3t	1	3	1t
Software management K.6.3	9t	-	-	1t	-	-	2t	-	-
Structured programming D.2.2	11	-	-	15	-	-	14t	-	-
Testing and debugging D.2.5	7	10	12	4t	15	11t	8t	15	8t
Tools and techniques D.2.2	5	2	3	7t	3	1	2t	2	4
User interfaces D.2.2	1	1	4	11t	5	8t	5t	9t	5
User interfaces H.5.2	#	#	14	#	#	8t	#	#	12t
User/machine systems H.1.2	-	11	2	-	10t	-	-	9t	-
Windows D.2.2	-	-	1	-	-	5t	-	-	1t

# = Node not in CCS for period.  
t = Tie for ranked position. Ties for position n all ranked as n; next ranked position begins at n+m, where m is number of ties ranked at n.  
- = Not in highest 15 for period.

Some common themes also emerge from the descriptor data. The following themes appear consistently and repeatedly: tools and techniques, user interfaces, programming environ-

ments, reusable software, design methodologies, software management and development, testing and debugging, verification, metrics, Ada, and requirements/specifications. Some new descriptors are prominent in 1991-1994 data, including computer-aided software engineering, object-oriented programming, and Windows.

Only the following 25 descriptors appeared in all time periods (not just among the 15 most common by period).<sup>13</sup>

- Ada D.3.2
- Concurrent programming D.1.3
- Curriculum K.3.2
- Design D.2.10
- General D.2.0
- Human factors H.1.2
- Interaction techniques I.3.6
- Introductory and survey A.1
- Management D.2.9
- Mathematical software G.4
- Methodologies D.2.1
- Metrics D.2.8
- Program verification D.2.4
- Programming environments D.2.6
- Requirements/specifications D.2.1
- Software development K.6.3
- Software libraries D.2.2
- Software management K.6.3
- Software quality assurance (sqa) D.2.9
- Specification techniques F.3.1
- Specifying and verifying and reasoning about programs F.3.1
- Testing and debugging D.2.5
- Tools and techniques D.2.2
- User interfaces D.2.2
- User/machine systems H.1.2

Another way to see the filtering effect of the algorithm is to count the descriptors in each major

---

<sup>13</sup>. Recall that new CCS descriptors created in 1987 and 1991 are not candidates for appearance in preceding time periods. Hence, some descriptors that are now commonly used, such as object-oriented programming D.1.5, could not appear in this list.

CCS category in the original data and compare that number to the ones that emerged as network nodes. Table 14 gives the percentages of the 57,727 descriptors in the original data by CCS category (first column), the percentages of the 1,606 unique descriptors in the original data by CCS category (second column), and the percentages by CCS category of the 158 descriptors that passed the co-word analysis filter to reach the resulting 42 networks.

**Table 14: CCS Descriptor Summary Data**

<b>CCS Category</b>	<b>All Descriptors (57,725)</b>	<b>Unique Descriptors (1,606)</b>	<b>Network Descriptors (158)</b>
A-General Literature	0.6%	0.4%	1.3%
B-Hardware	1.1%	7.5%	0%
C-Computer Systems Organization	4.4%	8.5%	3.2%
D-Software	59.1% (40.1%, in D.2)	31.9% (11.2% in D.2)	58.3% (29.1% in D.2)
E-Data	0.6%	1.8%	0%
F-Theory of Computation	4.5%	5.2%	6.3%
G-Mathematics of Computing	1.7%	5.3%	1.9%
H-Information Systems	8.9%	11.6%	8.9%
I-Computing Methodologies	7.6%	16.1%	12.0%
J-Computer Applications	2.5%	3.6%	1.3%
K-Computing Milieux	8.9%	8.1%	11.3%

The hardware and data CCS categories were not represented at all in the networks; and the general literature, computer systems organization, mathematics of computing, and computer applications categories were only marginally included. The theory of computation category was included primarily with respect to program verification.

Listed below are the 8 non-D.2 descriptors included among the 15 most frequent descriptors in networks (Table 13). These descriptors highlight interactions among D.2 descriptors and other descriptors in CCS: D.1.5 - Object-Oriented Programming.

D.3.2 - Ada

H.1.2 - Human factors

H.1.2 - User/machine systems

H.5.2 - User interfaces

I.2.1 - Applications and expert systems

K.6.3 - Software development

K.6.3 - Software management

## 8.2 Findings

Based on our analyses, it appears that much of software engineering's intersection with the rest of computing is in the areas of user interaction, software management, and programming methodology. Very little interaction with hardware, data, mathematics of computing, and computer applications is evident. Further analyses will reinforce this hypothesis.

Just as important, our analysis of the CCS descriptors shows that some D.2 descriptors play a less important role than implied in several software engineering definitions [Naur 69], [Boehm 76], [Zelkowitz 78], [Fairley 85], [Humphrey 89], [Shaw 90], [Denning 92], [IEEE 89]. These definitions normally incorporate terms such as large-scale, economical, managerial, interdisciplinary, production, maintenance, reliable, dependable, efficient, safety, design, and specifications. We see some of these themes in our findings, but not all of them.

Human factors is a consistent and important theme in all periods we analyzed. This is contrary to other attempts to define software engineering where human factors is often deemed marginal. Conversely, economic aspects are mentioned consistently in these other discussions. However, we found little on that subject in the research and development literature, even though descriptors under (D.2.9) Management - Cost Estimation, and Management - Time Estimation, as well as (K.6.0) General, Economics were available. Over the three time periods analyzed here, these descriptors appeared in the unfiltered data 117 times, 15 times, and 33 times, respectively, but did not associate strongly enough with other descriptors to be placed in any networks.

Also, we find little evidence of a maturing profession as judged by commentary on issues such as ethics, licensing, certification, human safety, and codes of good practice, even though appropriate CCS nodes are defined. None of these nodes reached the networks, and only minimal inclusion was found in the almost 58,000 total, unfiltered descriptors. While the standards descriptors were included in the first two generations of networks (but in isolated fashions), they did not appear in 1991-1994 networks. As stated by Shaw [Shaw 90], an engineering discipline of software is still in the early stages of development.



## 9 Conclusions

### 9.1 Methodology

This study demonstrates the feasibility of co-word analysis as a viable approach for extracting patterns from and identifying trends in large corpora where the texts collected are from the same subdomain and divided into roughly equivalent quantities for different time periods. This methodology has also been used in other studies at the Software Engineering Institute as a way of filtering risk information collected at external sites [Monarch 95] and for differentiating process assessments of external sites—those that showed an improvement from those that did not—with respect to thematic concerns. Moreover, the Software Engineering Risk Repository (SERR), an information retrieval system containing risk and risk mitigation information from over 35 software risk assessments, uses term co-occurrence networks for suggesting related terms to those found in a user's query [Monarch 96]. The system is currently being user tested.

### 9.2 Findings

What can we conclude about the state of software engineering based on our study of publications? First, the field is rapidly evolving as is demonstrated by the changing descriptors in networks, the changing connections in super networks, and the changing centrality/density scores. The analysis of the 1991-1994 data shows a trend towards focusing on object-oriented themes, software reuse/software library themes, and user interface themes. Consistent themes are evident over the time periods studied, although contexts change. Some consistent themes are user interfaces, tools and techniques, verification and validation, software reuse, requirements and specifications, and design methodologies.

#### 9.2.1 The Role of Software Tools

The core themes of user interfaces and software development (with object-oriented methods) both display underlying principles (such as screen design, design methodologies, reusable software, and so forth) together with software tools that embody some of these underlying principles. These tools include X-Windows, Microsoft Windows, Ada, C++, and UNIX. CASE tools are prominent in software development networks, but names of specific CASE tools are not present. This observation suggests that the maturity of a software engineering subfield can be gauged by the maturity of relevant supporting tools. Earlier we observed that the languages Pascal, Basic, and Cobol dropped from the software engineering descriptors, along with programming-in-the-small issues such as structured programming. They were replaced by programming-in-the-large issues and by a different set of supporting tools appropriate for large-scale software development environments. As software engineering matures, we can expect to observe the names of other specific software tools and systems, and we may see new core areas emerge as supporting tools are refined.

Because CCS is a fixed taxonomy with periodic updates to descriptors, the role of implicit subject descriptors may be crucial in observing trends between and across updates to the classi-

fication system. Therefore, names of languages and systems as reflected in CCS descriptors provide numerous insights into observing a field's maturation.

### **9.2.2 Software Engineering and Computer Science**

What is the relationship between software engineering and computer science? We know of no comparable study of computer science terminology, so a comparison is difficult, but some observations are apparent. The latest detailed curriculum model for computing [Denning 89], listed the nine subareas of computing as algorithms and data structures, programming languages, architecture, numerical and symbolic computation, operating systems, software engineering and methodology, database and information retrieval, artificial intelligence and robotics, and human-computer communication. These areas are not meant to be independent, of course.

As shown by its descriptor networks, software engineering incorporates topics from most of these areas, but it stands alone in its emphasis on management, process, design, testing, specifications, and other fundamental engineering terms. It fits the fundamental engineering paradigm better than it fits the mathematics or experimental science paradigms [Denning 89].

Software engineering certainly draws from computer science theories, but it also depends heavily on theories from management, psychology, mathematics, and other related fields. We feel it is emerging as a discipline in computing rooted in computer science, but with its own character and content.

### **9.2.3 Limitations of This Study**

This study is based exclusively on refined publications, so it represents topics that are more developed than some others. Surely, there is much activity in cost/time estimation, management of programming teams, and other important but relatively immature areas. The lag time from the invention of software technology until its acceptance into common practice is estimated at 15-20 years [Redwine 84], so this gap is not surprising. Also while CCS provides the proper focus for this study, it may have limitations with respect to more detailed studies of software engineering trends because of its fixed taxonomy. Applying co-word analysis to author-defined descriptors, to abstracts, or to a document's text may reveal observations complementary to the ones we noted.

## 10 References

- [Boehm 76] Boehm, B. "Software Engineering." *IEEE Transactions on Computers C-25*, 12 (December 1976): 1226-1241.
- [Boehm 94] Boehm, B. "The IEEE-ACM Initiative on Software Engineering as a Profession." *Software Engineering Technical Council Newsletter* 13, 1, (September 1994): 1.
- [Brooks 87] Brooks, Jr., F. "No Silver Bullet: Essence and Accidents of Software Engineering" *Computer* 20, 4 (April 1987): 10-19.
- [Buckley 93] Buckley, F. "Defining Software Engineering." *Computer* 26, 8 (August 1993): 76-78.
- [CR 95] "Periodicals Received." *Computing Reviews* 36, 11 (November 1995): 599-608.
- [CR 96] "The Full Computing Reviews Classification System." *Computing Reviews* 37, 1 (January 1996): 4-16.
- [Callon 86] Callon, M.; Law, J.; & Rip, A. "Qualitative scientometrics." *Mapping of the Dynamics of Science and Technology*, London: McMillian, 1986.
- [Callon 91] Callon, M; Courtial, J-P.; & Laville, F. "Co-word Analysis as a Tool for describing the Network of Interactions between Basic and Technological Research: The Case of Polymer Chemistry." *Scientometrics* 22, 1 (January 1991): 153-203.
- [Coulter 91] Coulter, N. "Changes to the CR Classification System." *Computing Reviews* 32, 1(January 1991): 7-10.
- [Coulter 94] Coulter, N.; & Dammann, J. "Current Practices, Culture Changes, and Software Engineering Education." *Computer Science Education* 5, 2 (1994): 91-106.
- [Courtial 89] Courtial, J-P; & Law, J. "A Co-Word Study of Artificial Intelligence." 301-311. *Social Studies in Science*, London: SAGE, 1989.

- [Denning 89] Denning, P; Gries, D.; Mulder, M.; Tucker, A.; Turner, J.; & Young, P. "Computing as a Discipline." *Communications of the ACM* 32, 1 (January 1989): 9-23.
- [Denning 92] Denning, P. "Educating a New Engineer." *Communications of the ACM* 35, 12 (December 1992): 82-97.
- [Dijkstra 89] Dijkstra, E. "On the Cruelty of Really Teaching Computer Science." *Communications of the ACM* 32, 12 (December 1989): 1398-1404.
- [IEEE 89] *Software Engineering Standards*. New York: Institute of Electrical and Electronics Engineering, Inc. 1989.
- [Fairley 85] Fairley, R. *Software Engineering Concepts*. New York: McGraw-Hill, 1985.
- [Ford 89] Ford, G.; & Gibbs, N. "A Master of Software Engineering Curriculum." *Computer* 22, 9 (September 1989): 59-71.
- [Gibbs 89] Gibbs, N. "The SEI Education Program: The Challenge of Teaching Future Software Engineers." *Communications of the ACM* 32, 5 (May 1989): 594-605.
- [Gibbs 91] Gibbs, N. "Software Engineering and Computer Science: The Impending Split." *Education and Computing* 7 (1991): 111-117.
- [Gries 91] Gries, D. "Teaching Calculation and Discrimination: A More Effective Curriculum." *Communications of the ACM* 34, 3 (March 1991): 45-54.
- [Humphrey 89] Humphrey, W. *Managing the Software Process*. Reading, Mass: Addison-Wesley, 1989.
- [Jackson 94] Jackson, M. "Problems, Methods, and Specialization." *Software* 11, 6 (November 1994): 57-62.
- [Law 92] Law, J; & Whittaker, J. "Mapping Acidification Research: A Test of the Co-Word Method." *Scientometrics* 23, 3 (1992): 417-461.

- [Monarch 95] Monarch, I; & Gluch, D. *An Experiment in Software Development Risk Information Analysis* (CMU/SEI-95-TR-014, ADA 302320). Pittsburgh: PA: Software Engineering Institute, Carnegie Mellon University, October 1995.
- [Monarch 96] Monarch, I; Konda, S.; and Carr, M. "Software Engineering Risk Repository." *1996 Software Engineering Process Group (SEPG) Conference*. Atlantic City, New Jersey, May 20-23; 1996.
- [Naur 69] Naur, P.; & Randell, B (eds.), *Software Engineering: A Report on a Conference sponsored by the NATO Science Committee*. NATO, 1969.
- [Parnas 85] Parnas, D. "Software Aspects of Strategic Defense Systems." *Communications of the ACM* 28, 12 (December 1985): 1326-1335.
- [Parnas 90] Parnas, D. "Education for Computer Professionals." *Computer* 23, 1 (January 1990): 17-22.
- [Redwine 84] Redwine, S., Jr.; Becker, G; Marmor-Squires, A.; Martin, R.; Nash, S.; and Riddle, W. "DoD Related Software Technology Requirements, Practices, and Prospects for the Future." IDA Paper P-1788, Institute for Defense Analyses, Alexandria, Va. (June 1984).
- [Sammet 82] Sammet, J; & Ralston, A. "The New (1982) *Computing Reviews* Classification System-Final Version." *Communications of the ACM* 25, 1 (January 1982): 13-26.
- [Sammet 83] Sammet, J. "Summary of Changes from 1982 to 1983 Version of CR Classification System." *Computing Reviews* 24, 1 (January 1983): 7-8.
- [Sammet 87] Sammet, J. "Summary of Additions from 1983 to 1987 Version of CR Classification System." *Computing Reviews* 28, 1 (January 1987): 5-6.
- [Shaw 90] Shaw, M. "Prospects for an Engineering Discipline of Software." *Software* 6, 6 (November 1990): 15-24.
- [Turner 88] Turner, W.; Chartron, G.; Laville, F.; & Michelet, B. "Packaging Information for Peer Review: New Co-Word Analysis Techniques." A.

Van Raan (ed.), *Handbook of Quantitative Studies of Science and Technology*. Amsterdam: North-Holland, 1988.

[Whittaker 89] Whittaker, J. "Creativity and Conformity in Science: Titles, Keywords, and Co-Word Analysis." *Social Science in Science*, London: SAGE, 1989.

[Zelkowitz 78] Zelkowitz, M. "Perspectives on Software Engineering." *Computing Reviews* 10, 2, (June 1978): 197-216.

## Appendix: Maps of All Networks

Following are maps of all 42 networks generated by the co-word analysis used in this study. These images were captured directly from the output of a graphical user interface and are presented in that form. Corresponding maps in the body of the paper were reconstructed to enhance readability. To facilitate automatic processing of networks, CCS descriptors and node codes were appended in the original maps. In the following maps, nodes such as “metricsd2.8” should be interpreted as “Metrics D.2.8.”

Pass-1 descriptors are enclosed by thick boxes; while Pass-2 descriptors are enclosed by thin boxes. Pass-1 links are shown by thick lines, Pass-2 links are shown by thin lines. Hashed lines indicate two Pass-1 nodes linked during Pass-2; recall such links are treated as Pass-1 links because they join two Pass-1 nodes.

## A.1 1982-1986 Maps of 15 Networks

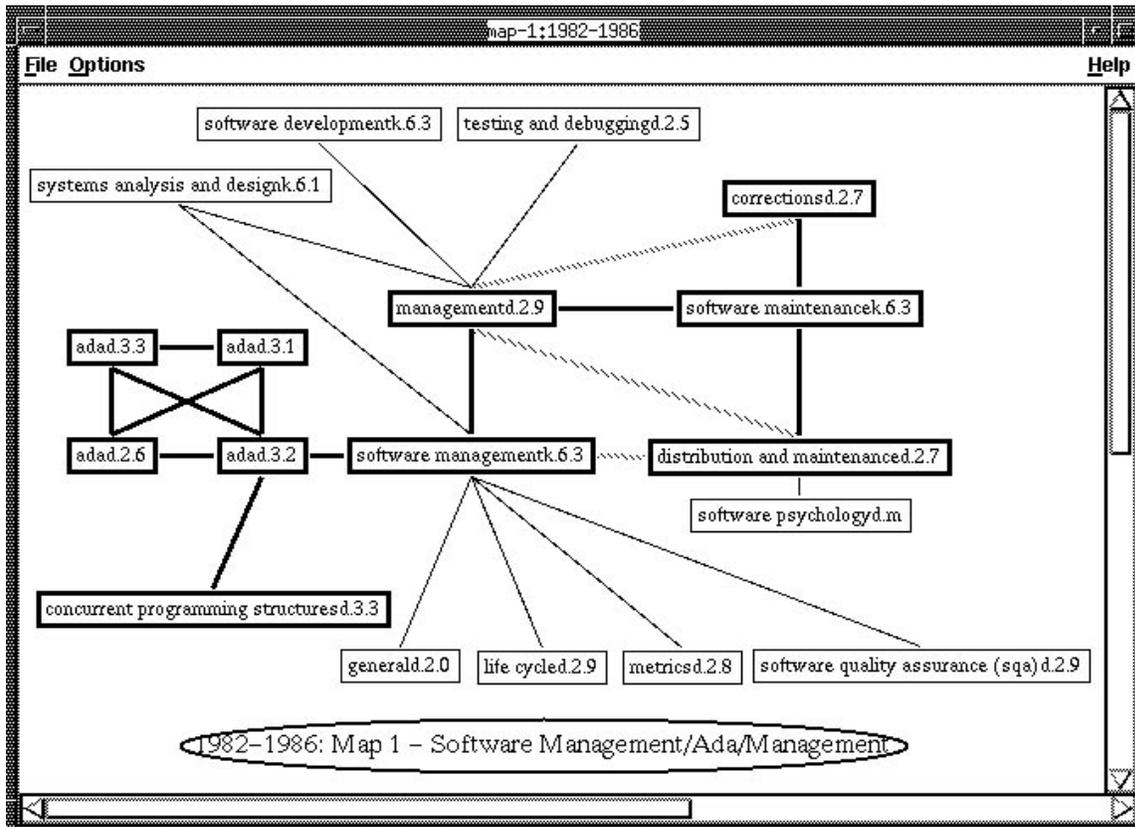


Figure A.1-1: Software Management - Ada

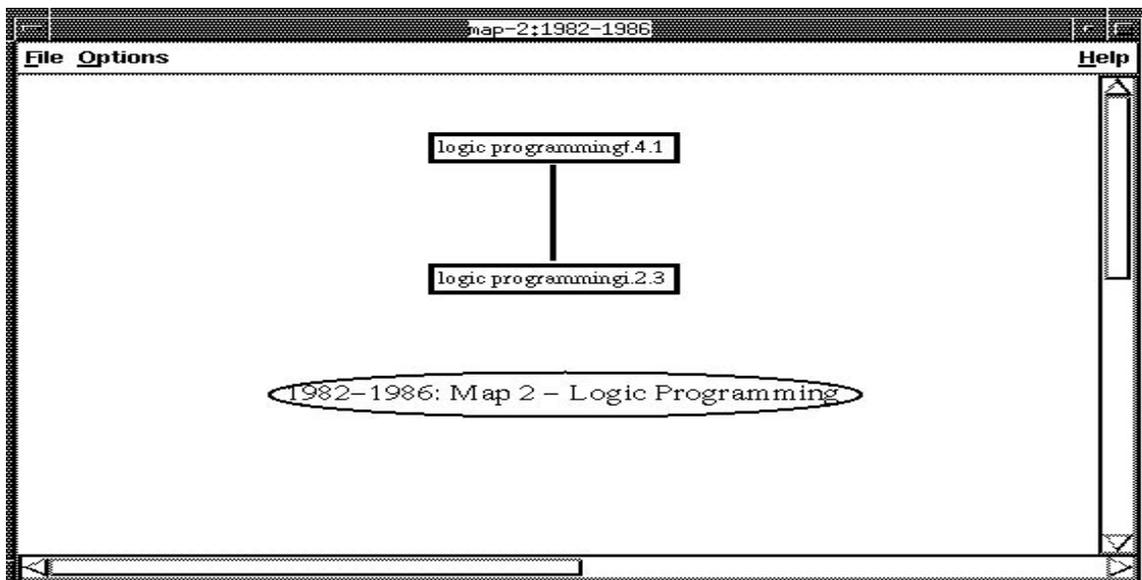


Figure A.1-2: Logic Programming

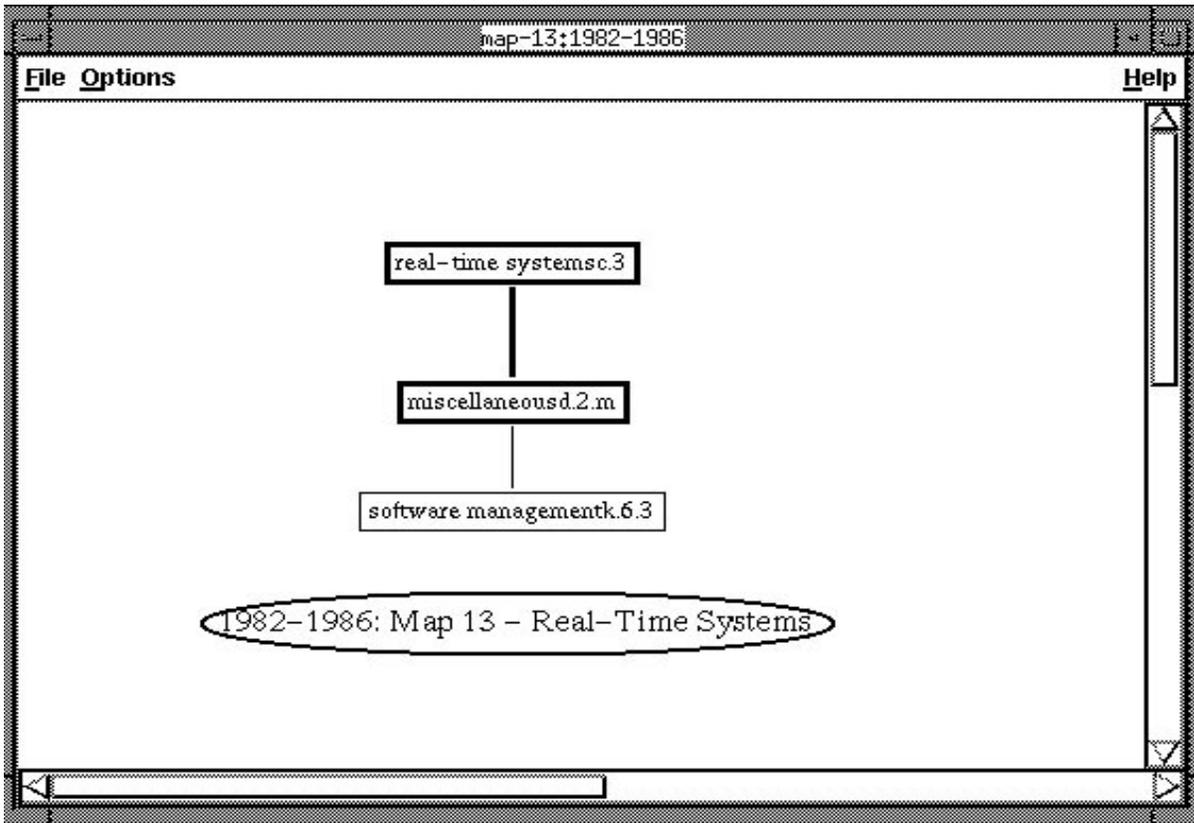


Figure A.1-3: User Interfaces

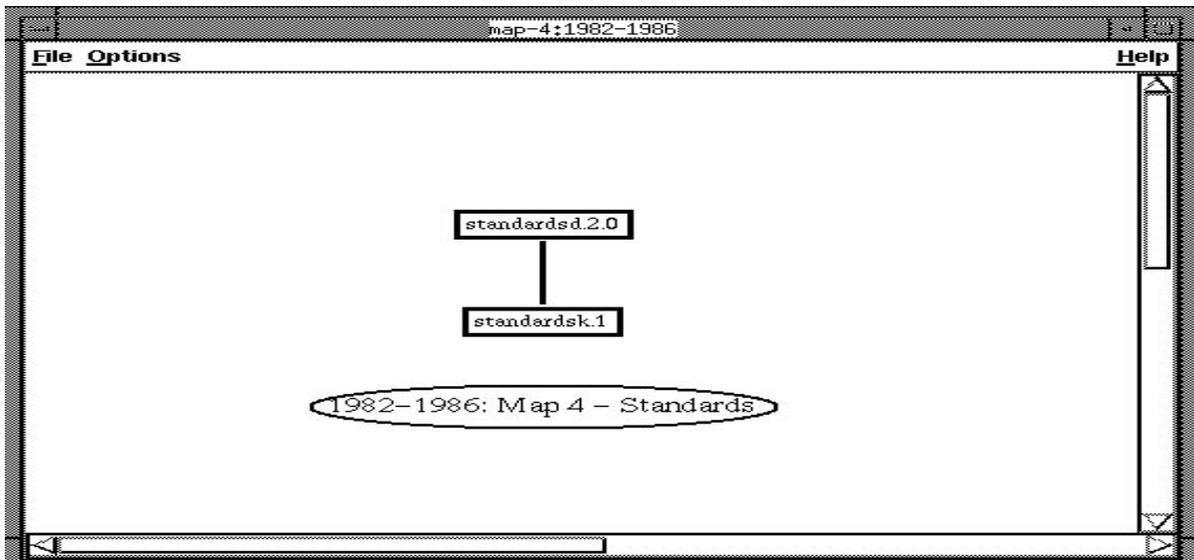


Figure A.1-4: Standards

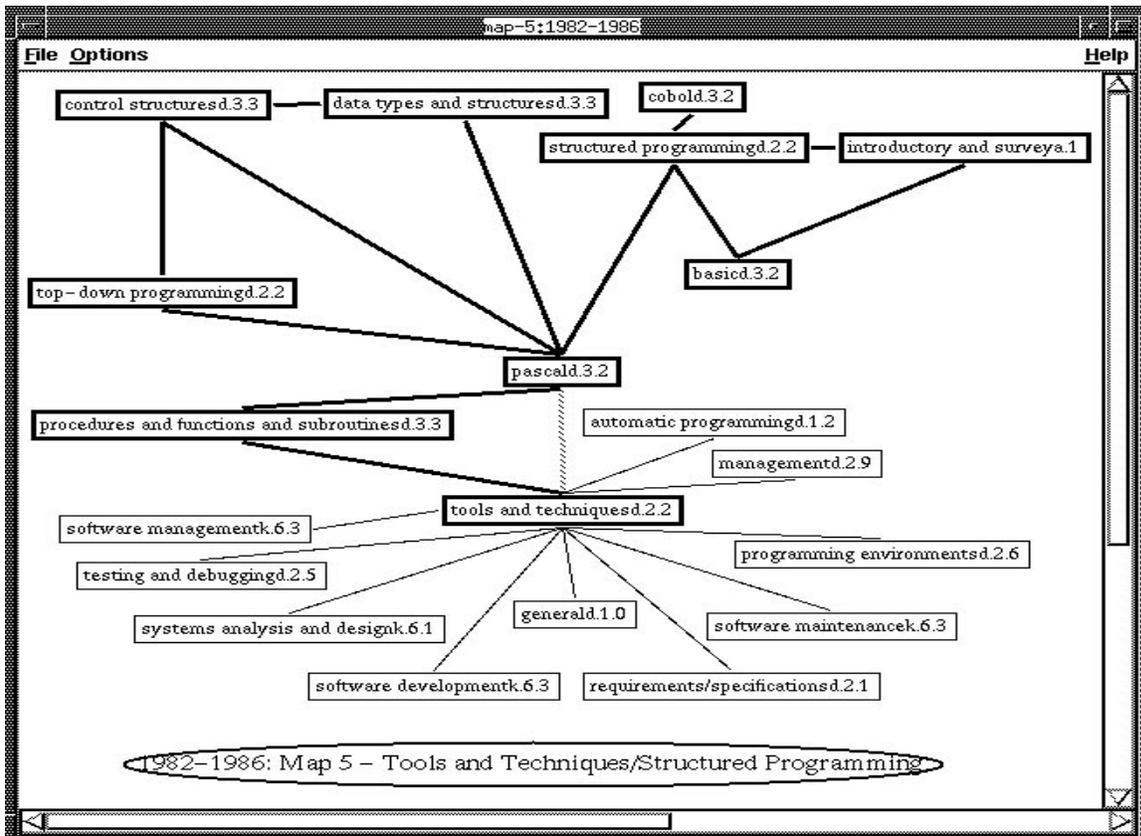


Figure A.1-5: Tools and Techniques - Structured Programming - Pascal

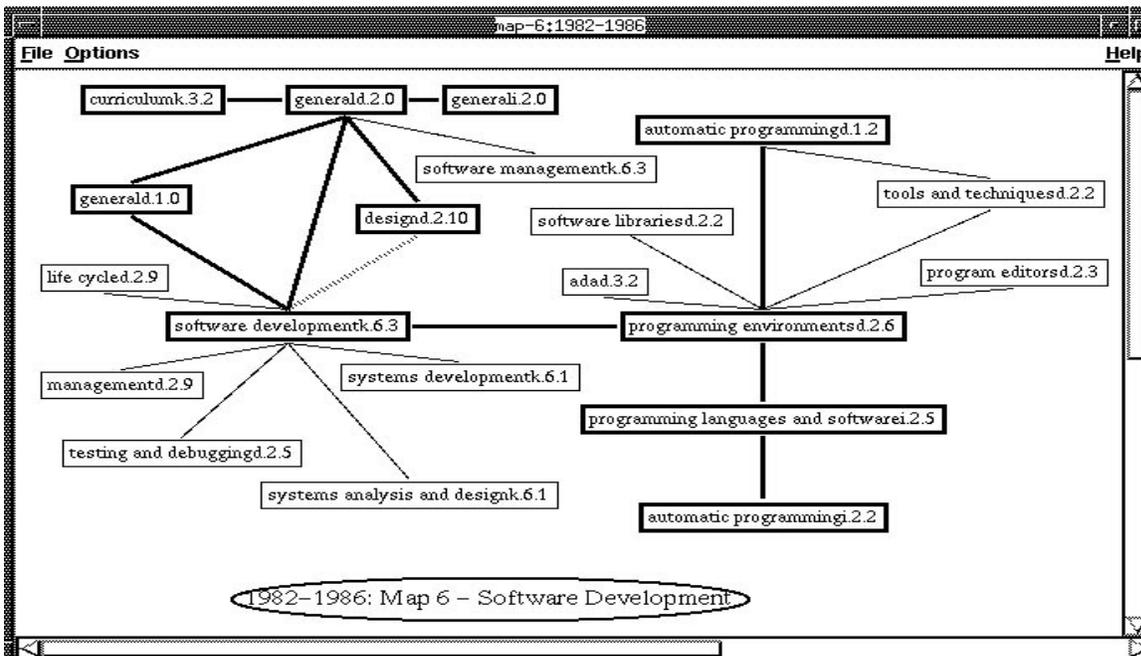


Figure A.1-6: Software Development

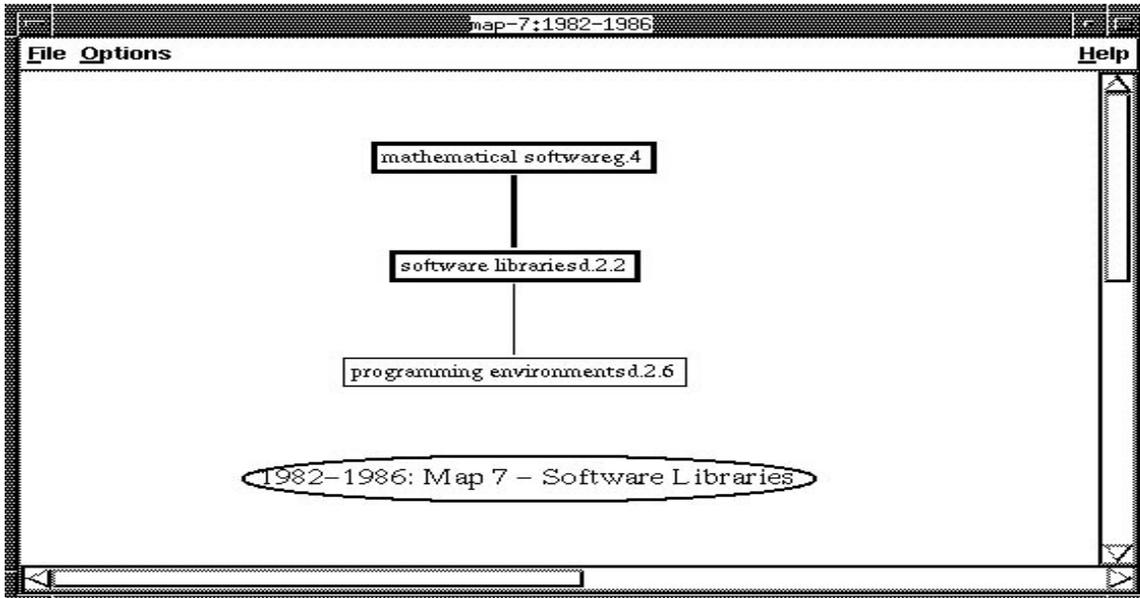


Figure A.1-7: Software Libraries

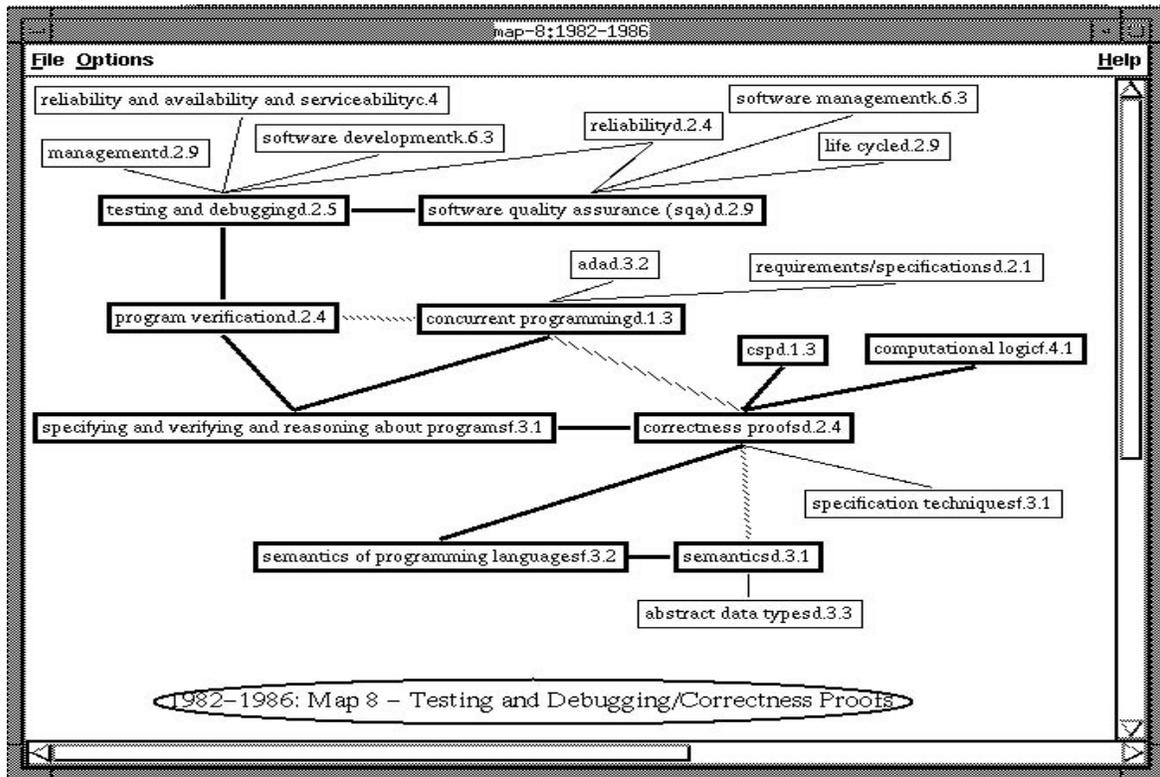


Figure A.1-8: Testing and Debugging - Correctness Proofs

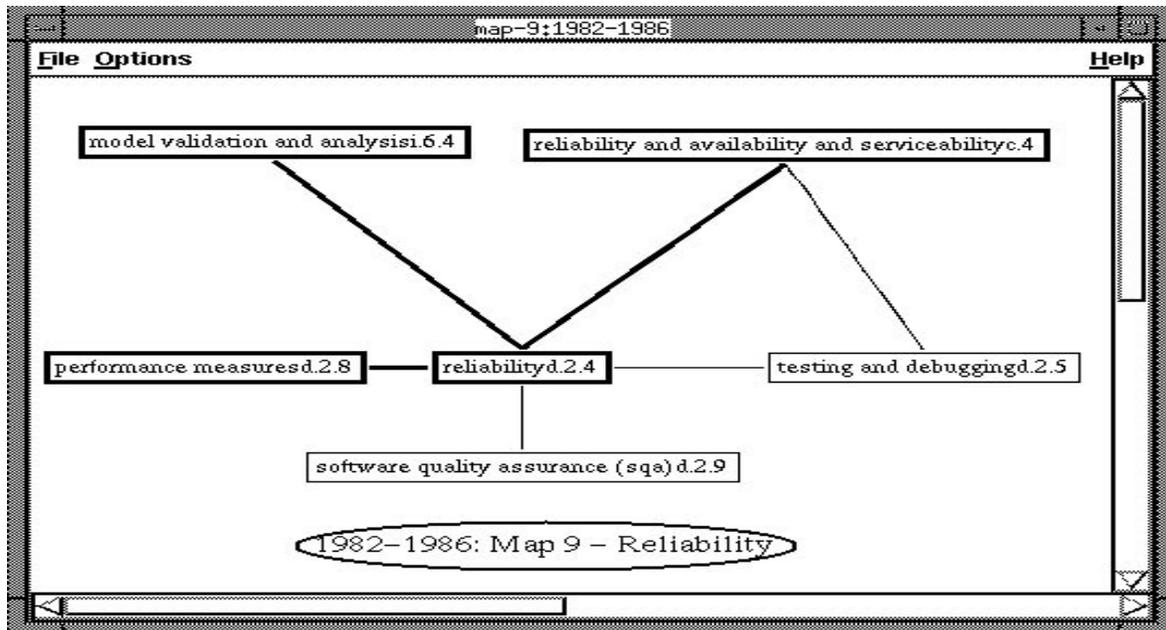


Figure A.1-9: Reliability

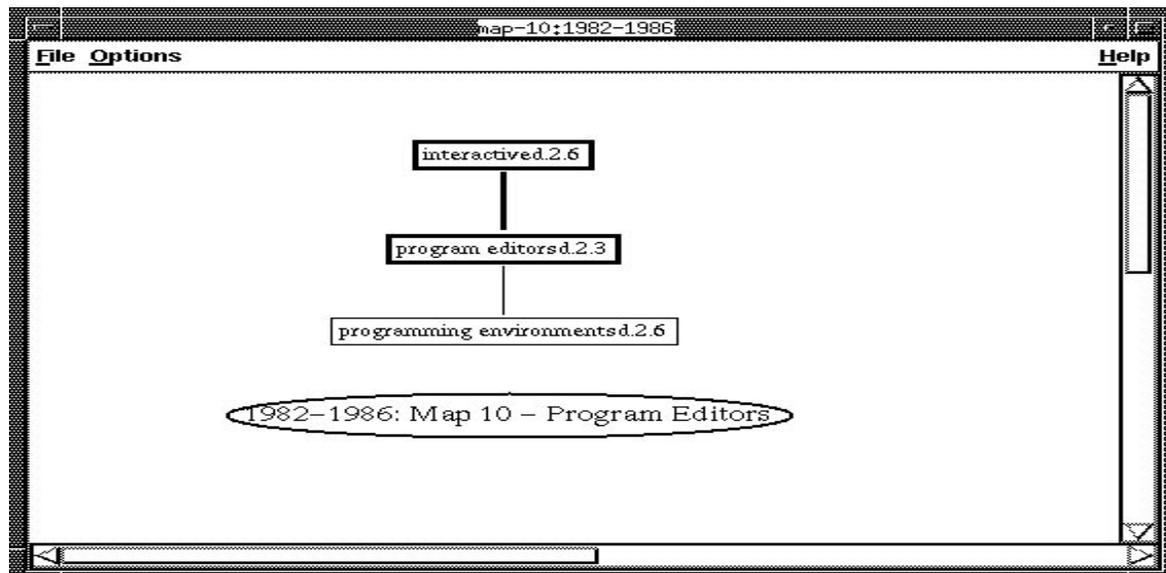


Figure A.1-10: Program Editors

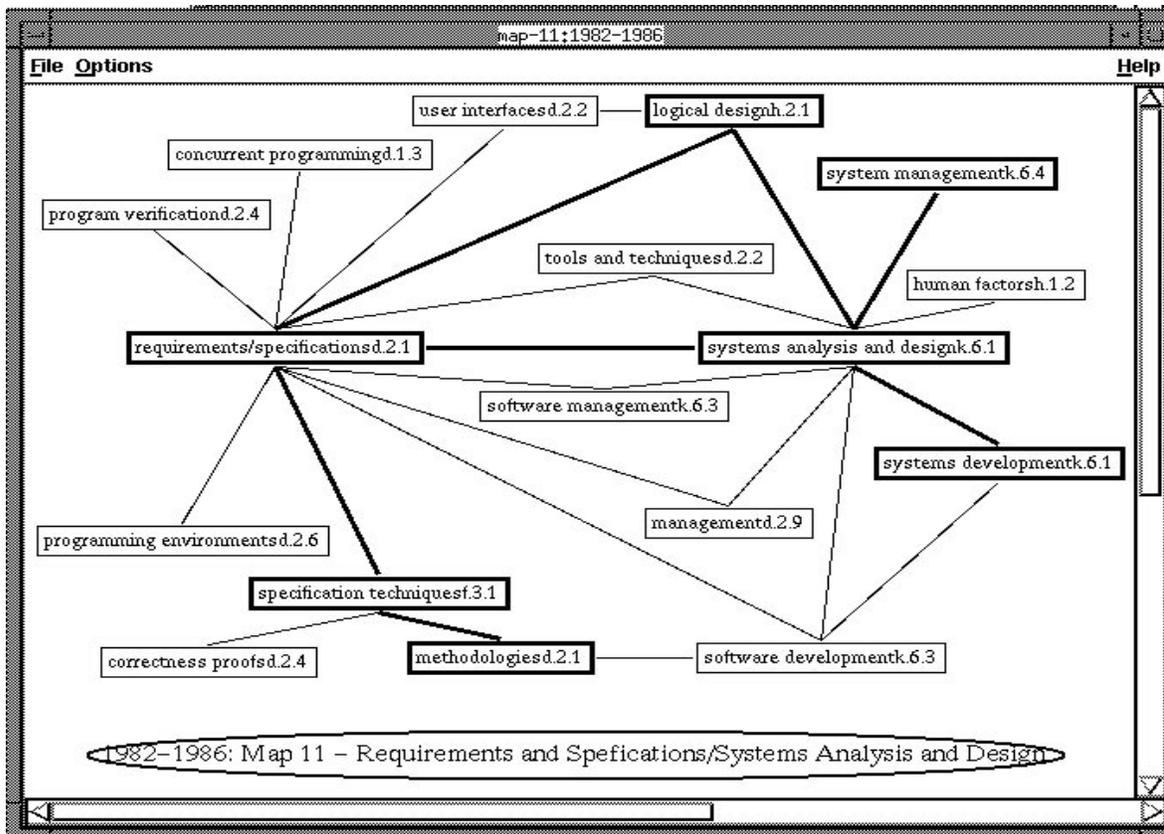


Figure A.1-11: Requirements/Specifications - Systems Analysis and Design

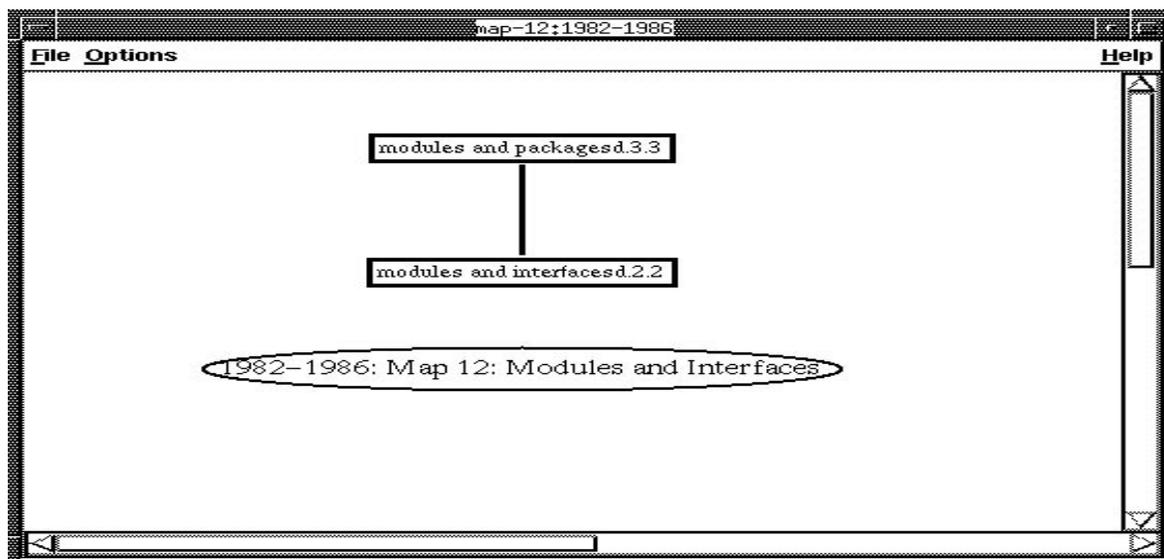


Figure A.1-12: Modules and Interfaces

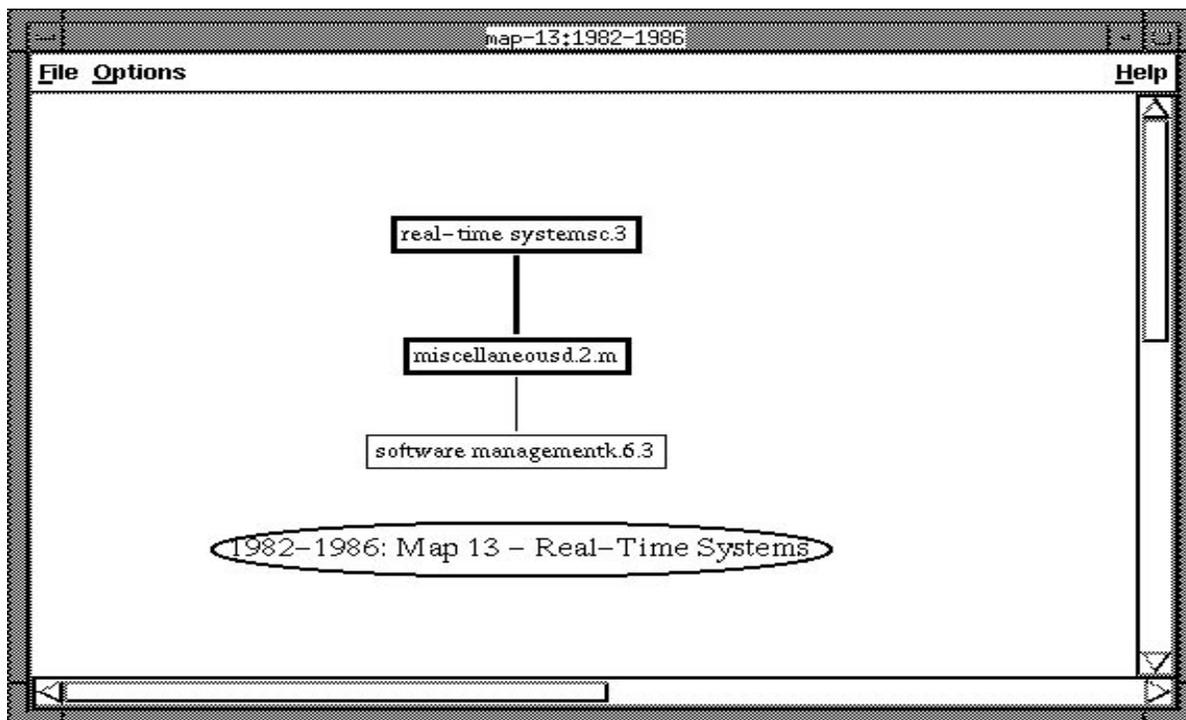


Figure A.1-13: Real-Time Systems

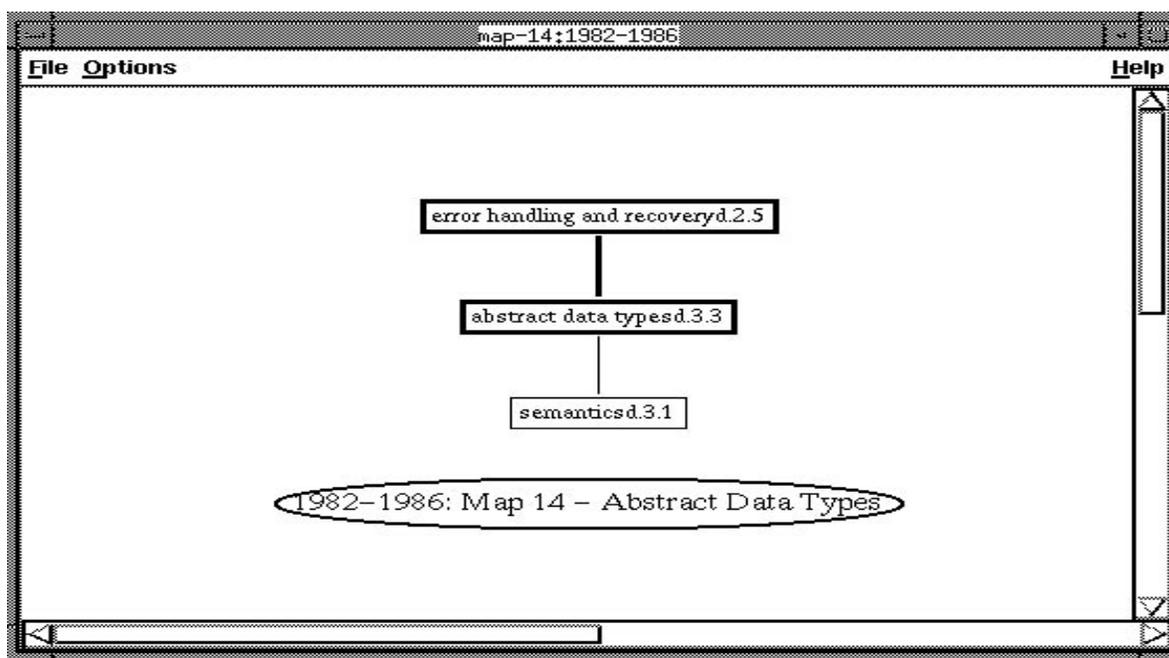


Figure A.1-14: Abstract Data Types

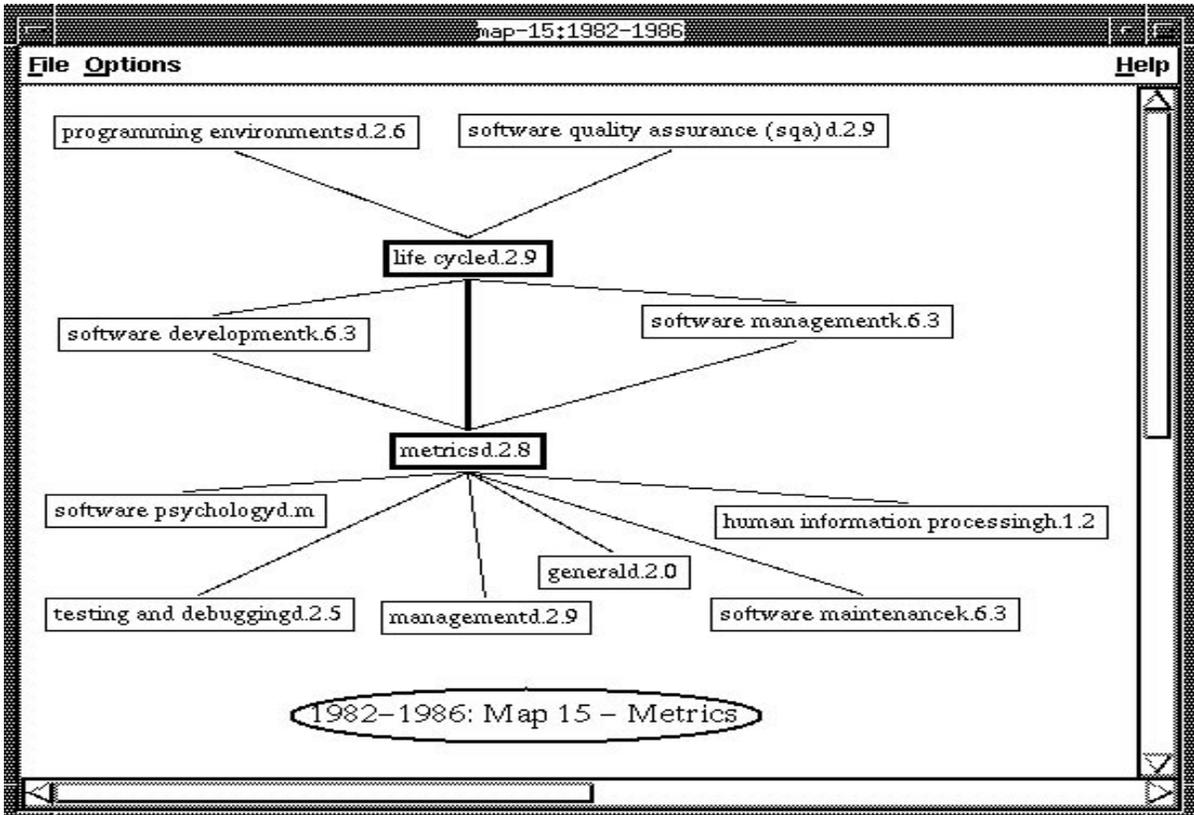


Figure A.1-15: Metrics

## A.2 1987-1990 Maps of 16 Networks

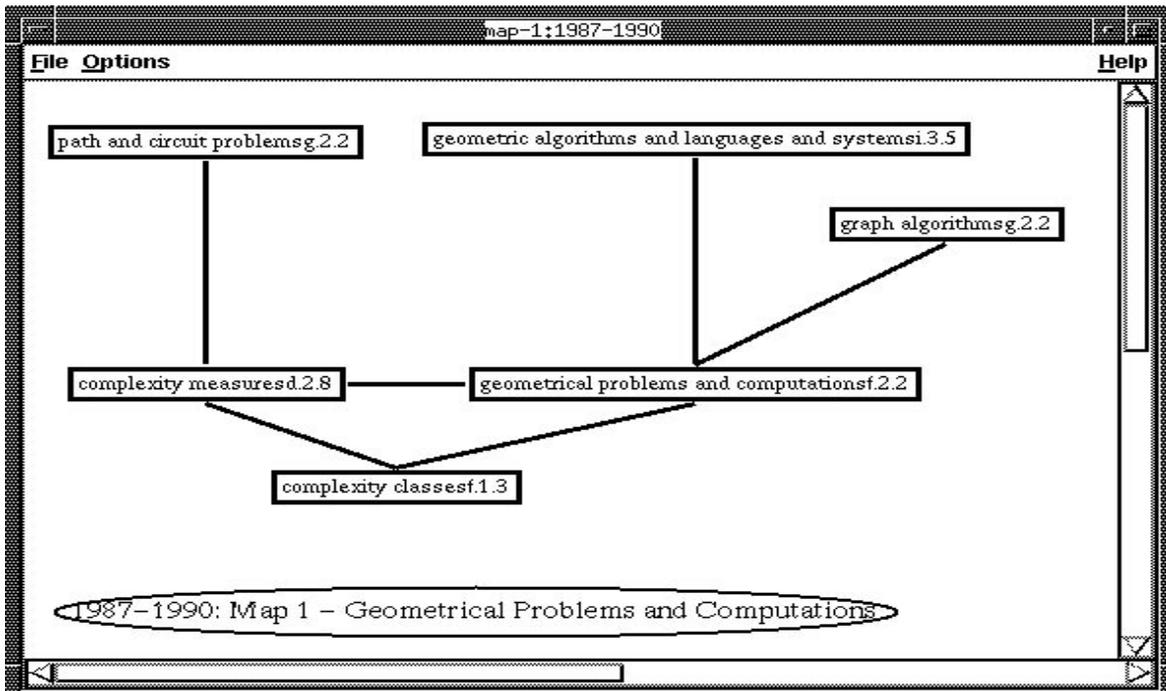


Figure A.2-1: Geometrical Problems and Computations

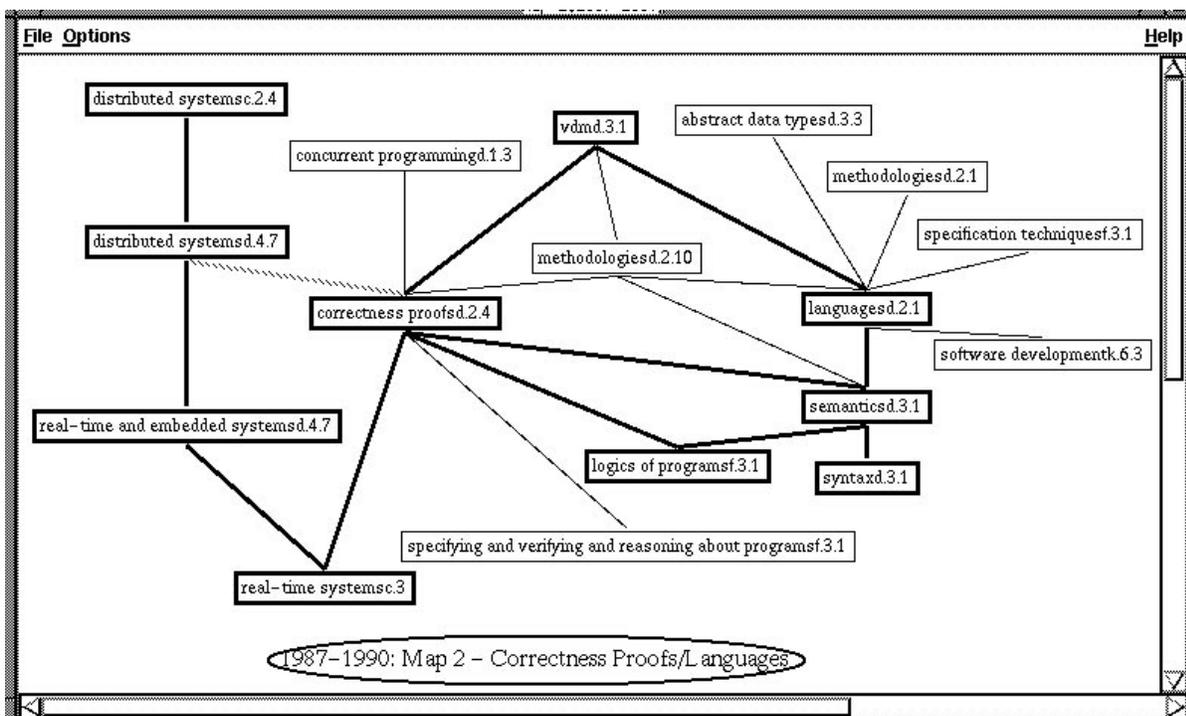


Figure A.2-2: Correctness Proofs - Languages



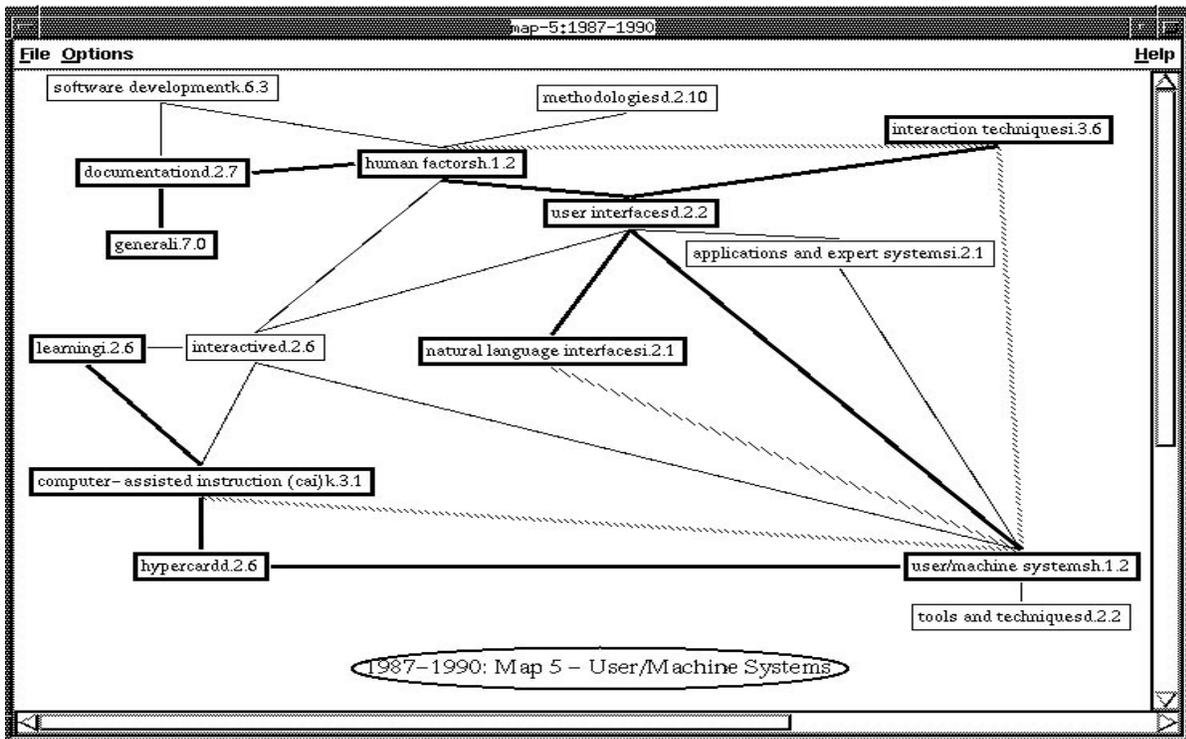


Figure A.2-5: User/Machine Systems

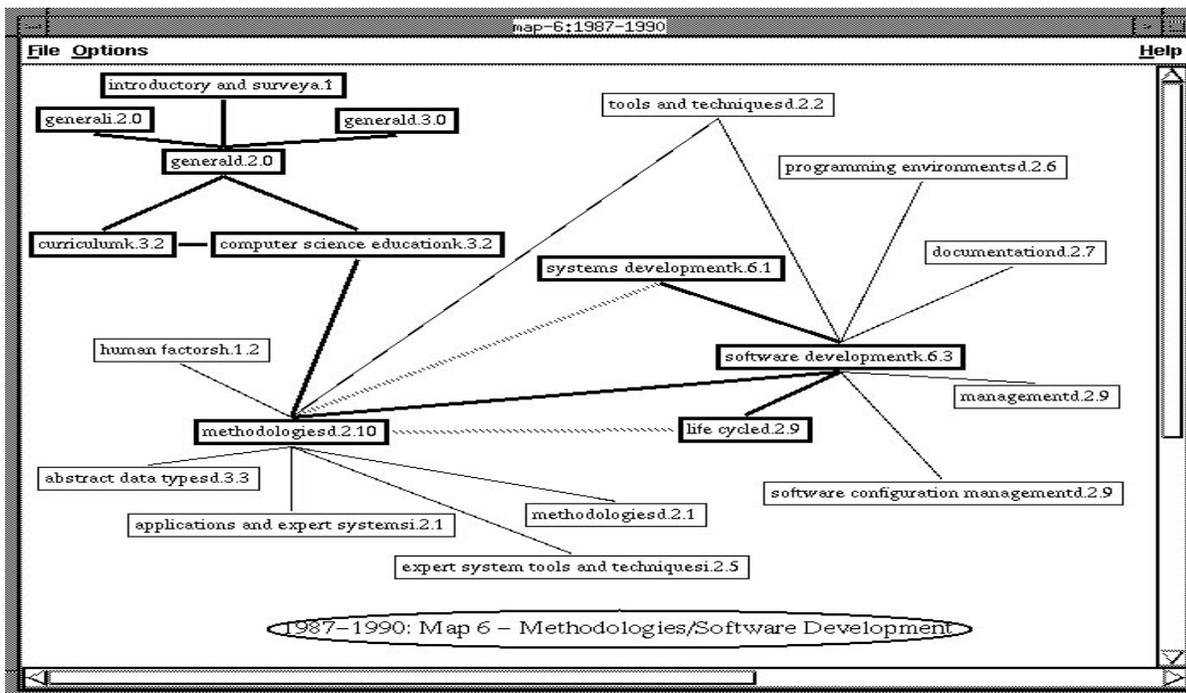


Figure A.2-6: Methodologies - Software Development

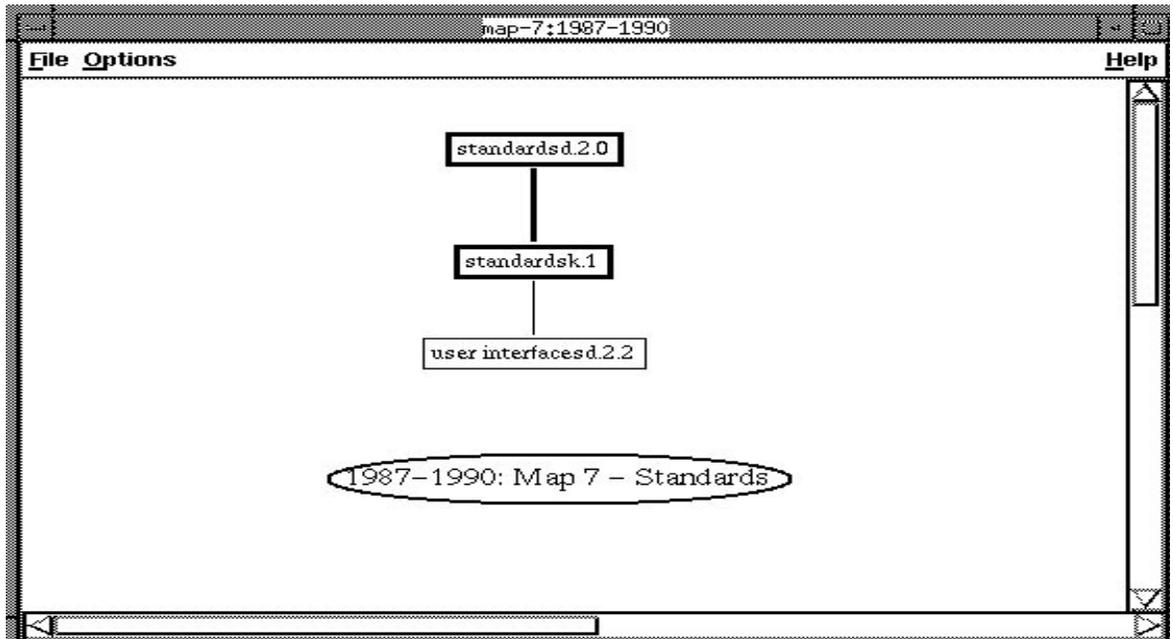


Figure A.2-7: Standards

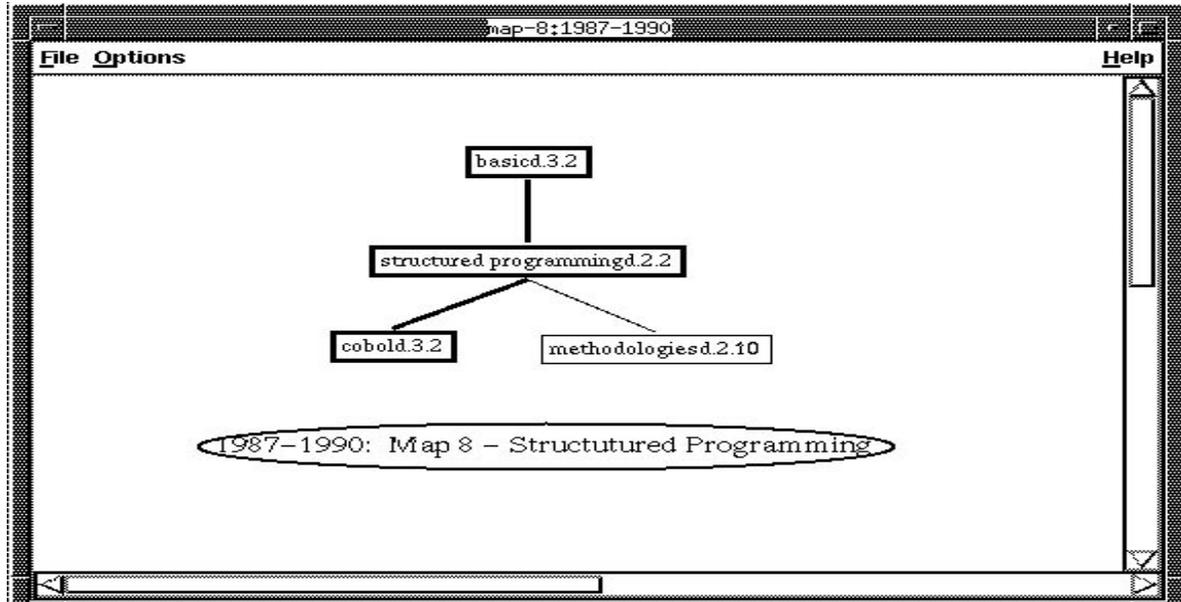


Figure A.2-8: Structured Programming

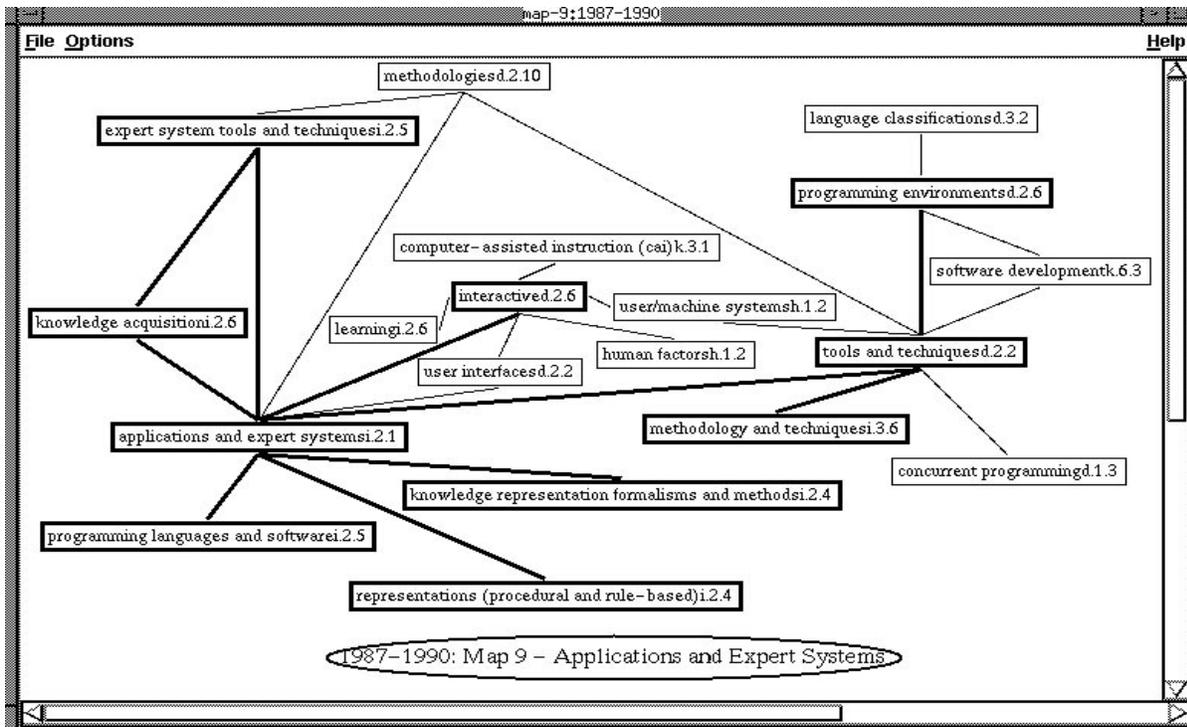


Figure A.2-9: Applications and Expert Systems - Tools and Techniques

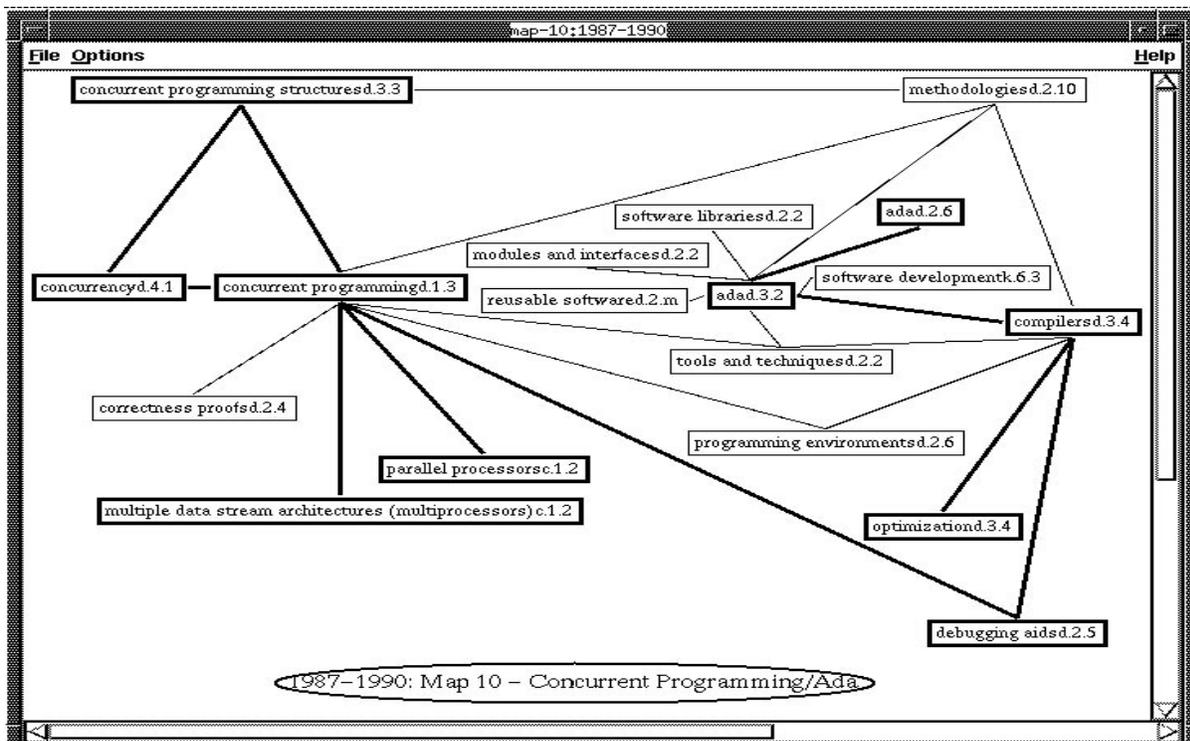


Figure A.2-10: Concurrent Programming - Ada

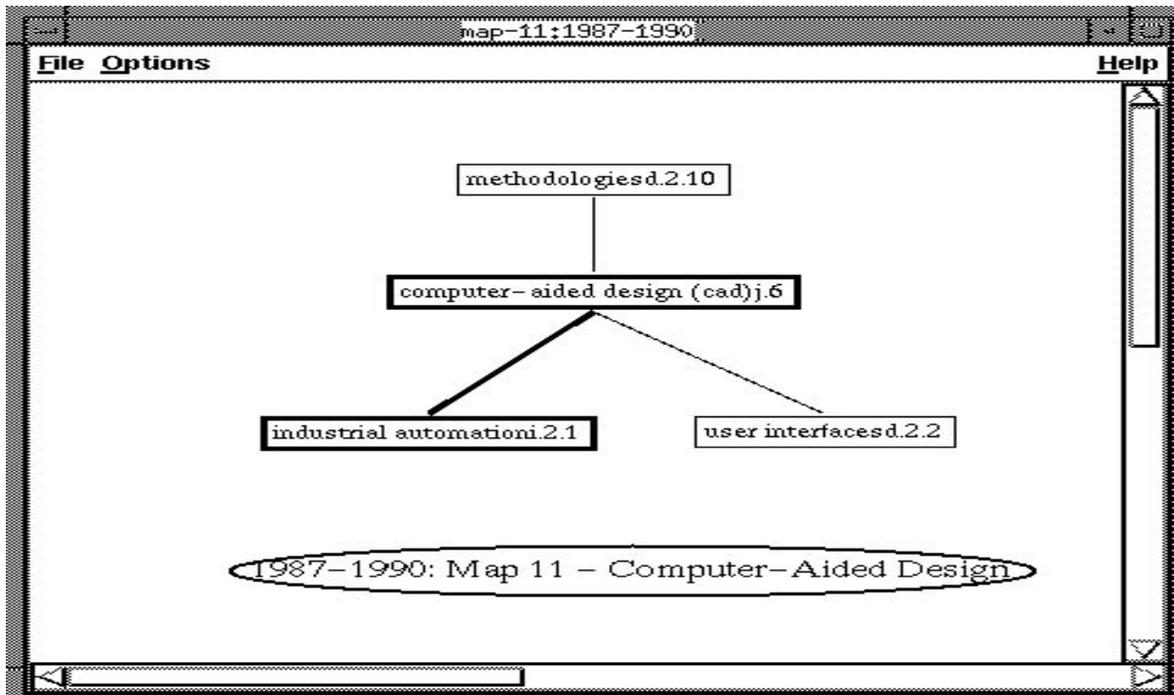


Figure A.2-11: Computer-Aided Design

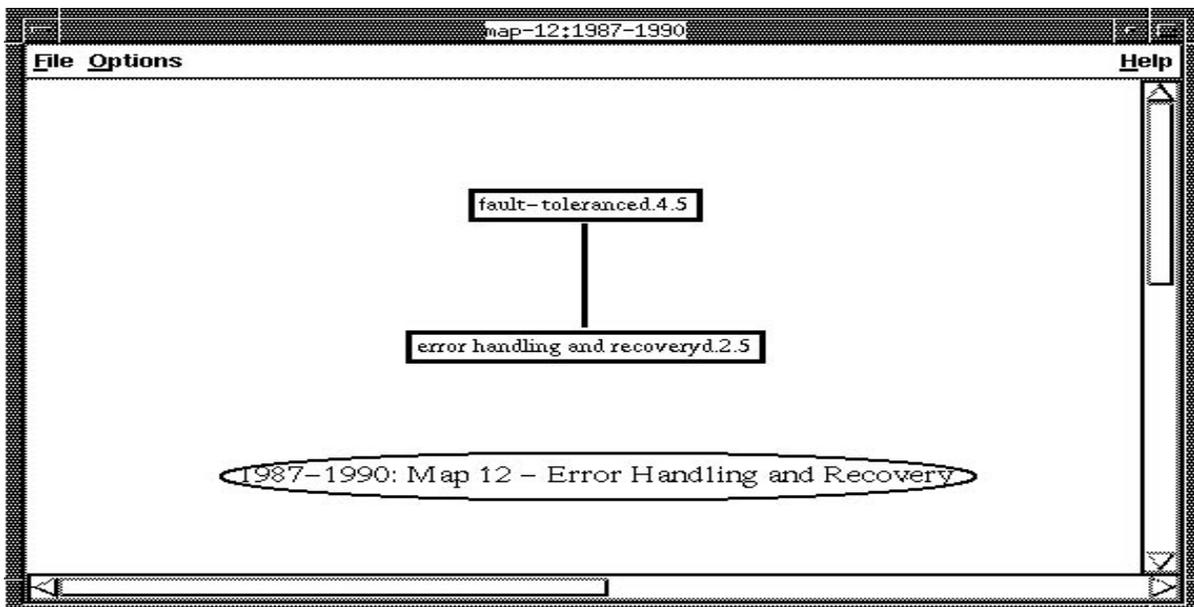


Figure A.2-12: Error Handling and Recovery

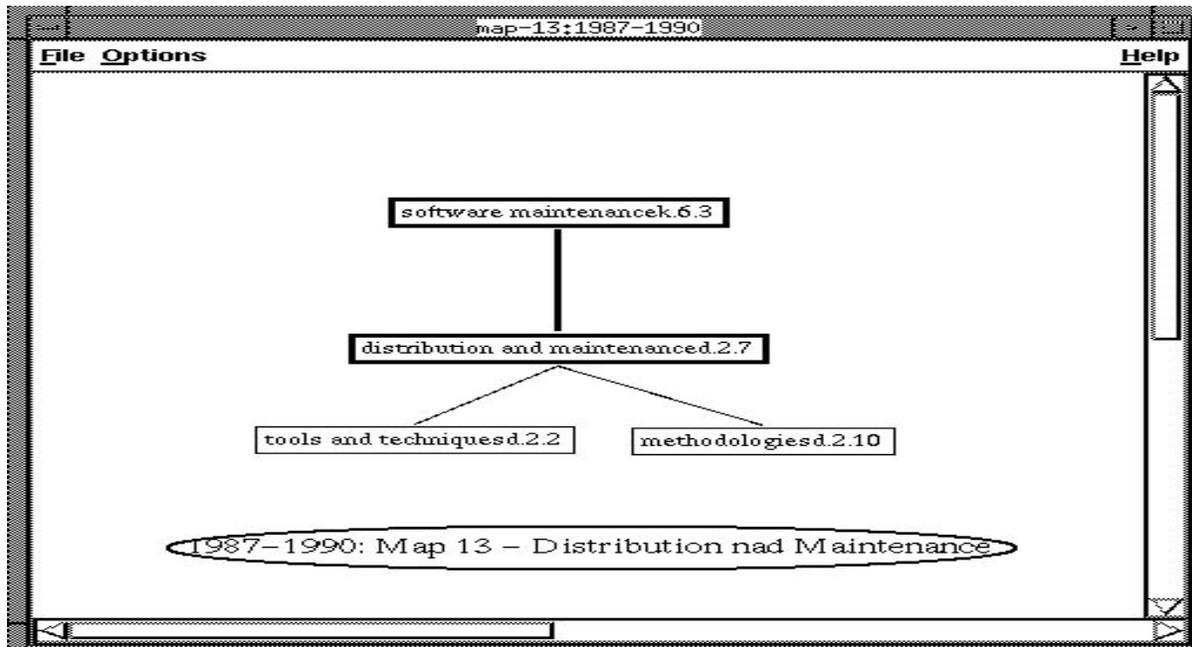


Figure A.2-13: Distribution and Maintenance

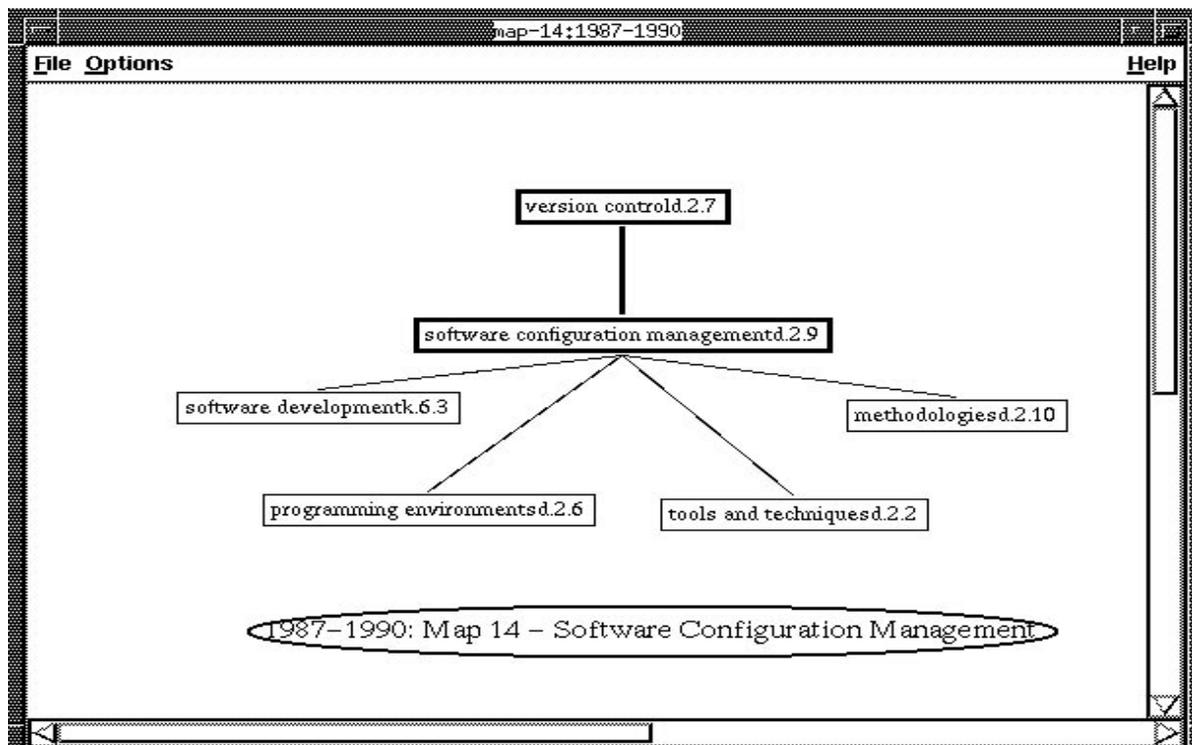


Figure A.2-14: Software Configuration Management

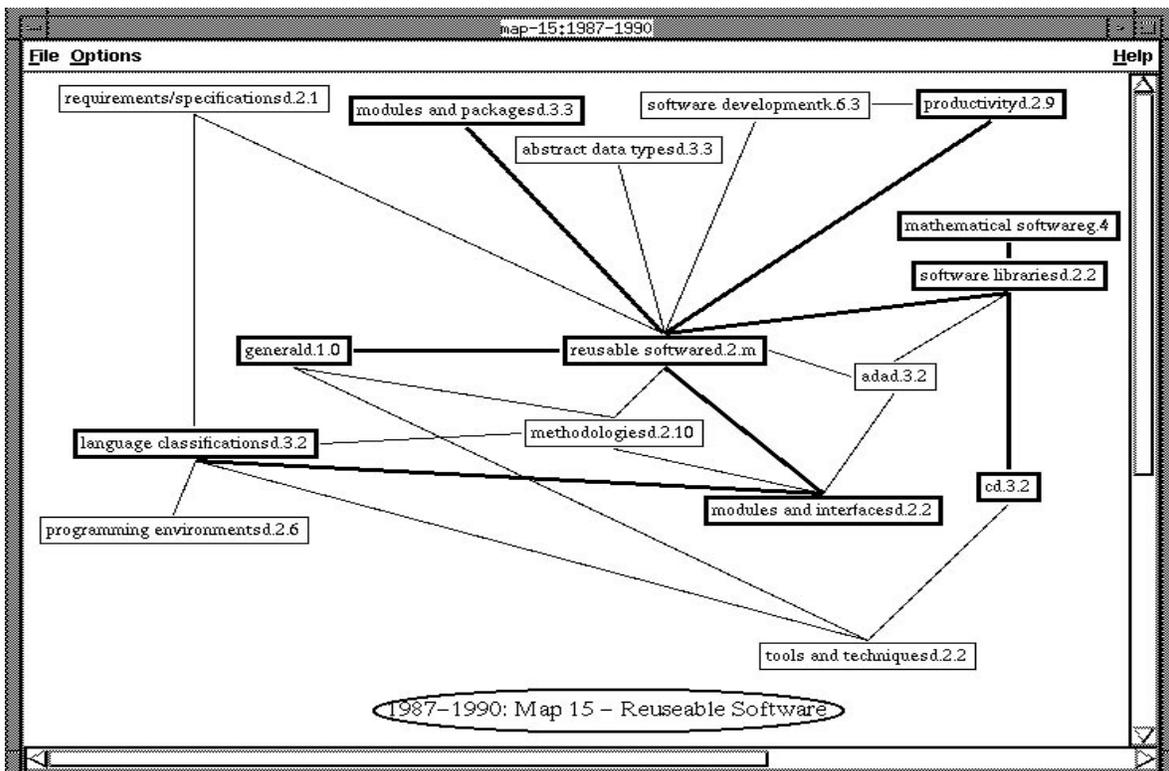


Figure A.2-15: Reusable Software

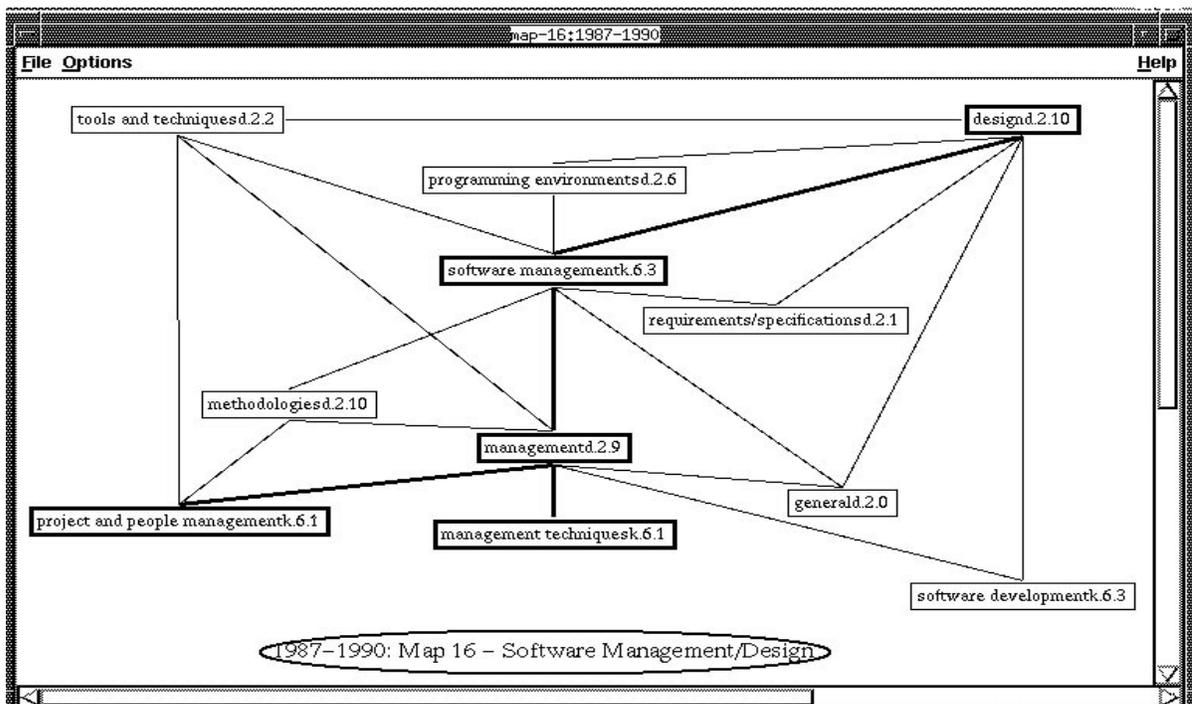


Figure A.2-16: Software Management - Design

### A.3 1991-1994 Maps of 11 Networks

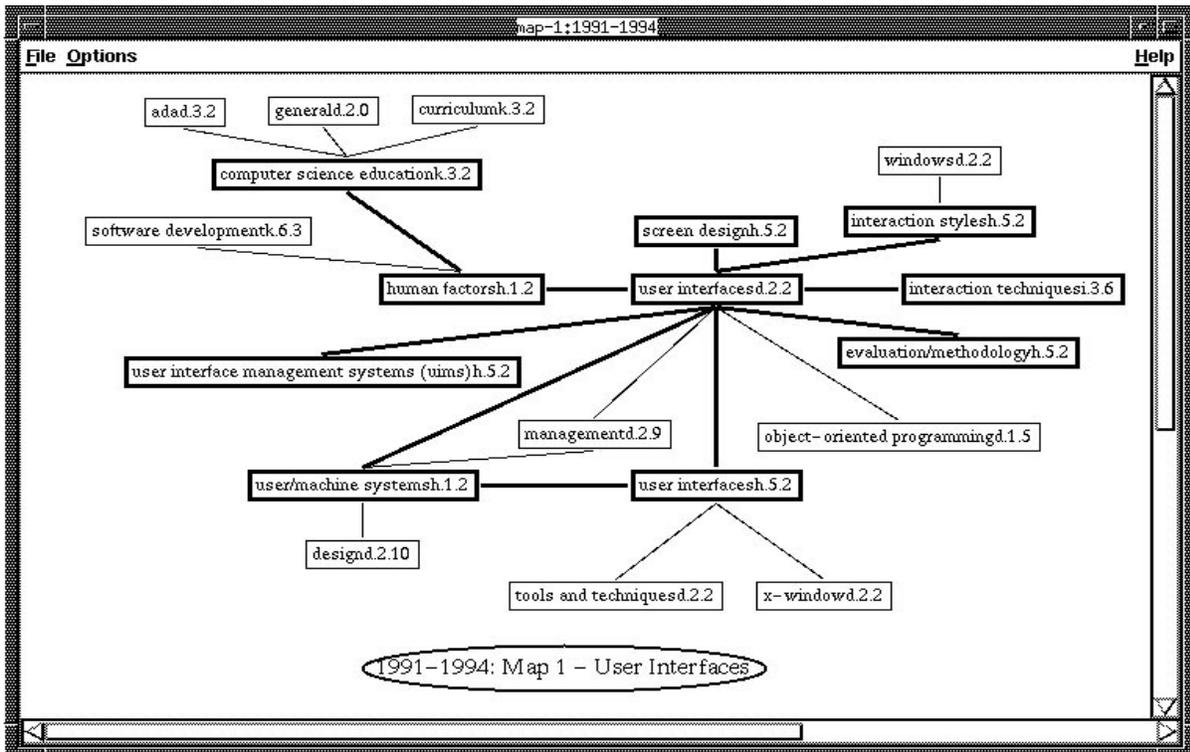


Figure A.3-1: User Interfaces

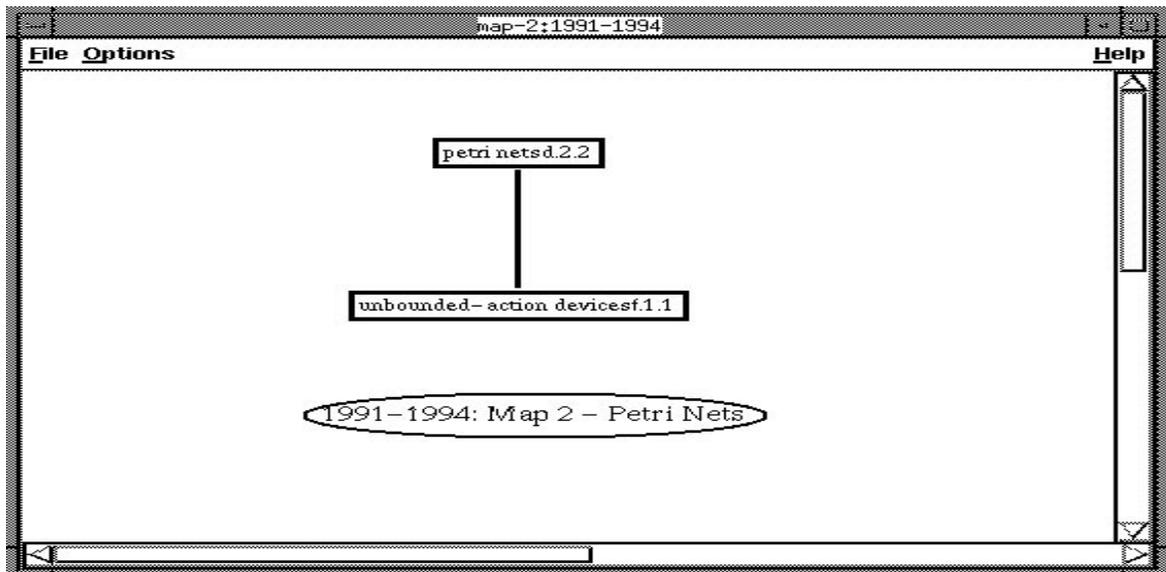


Figure A.3-2: Petri Nets

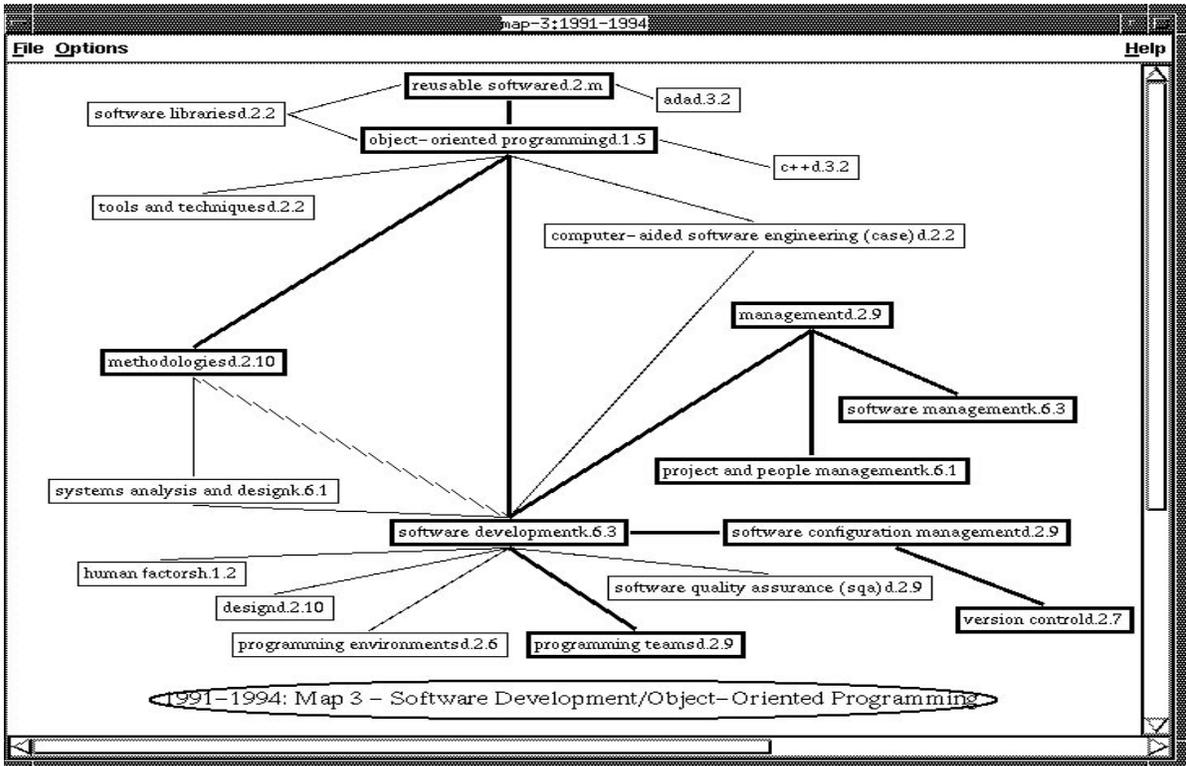


Figure A.3-3: Software Development - Object-Oriented Programming

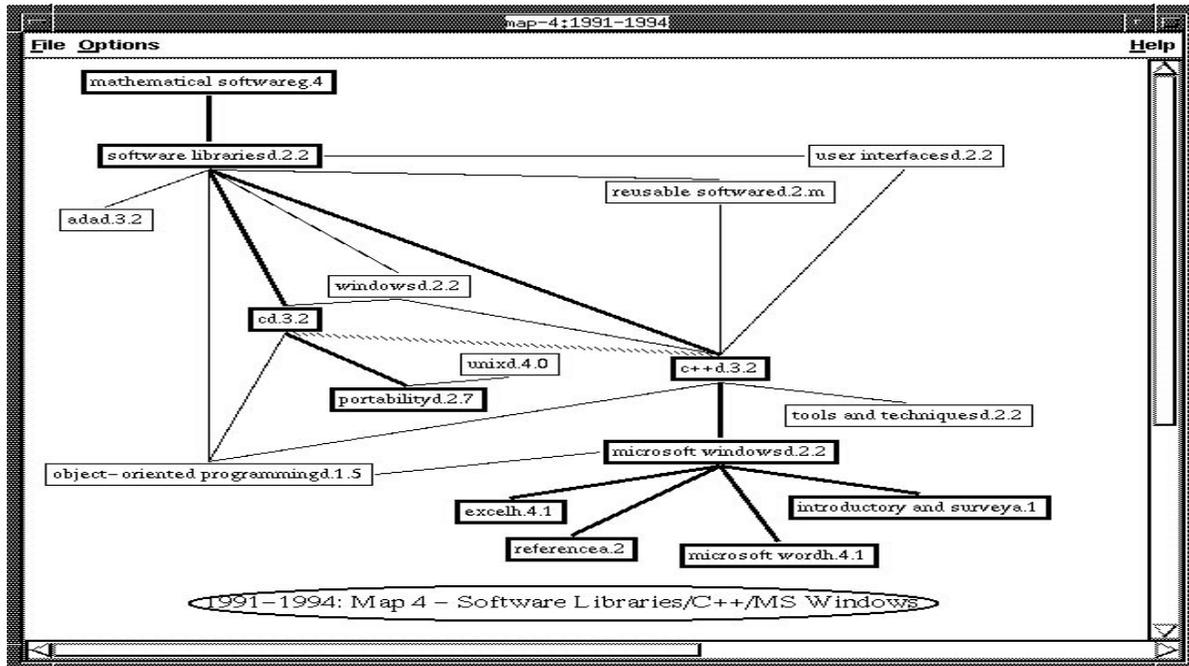


Figure A.3-4: Software Libraries - C++ - Microsoft Windows

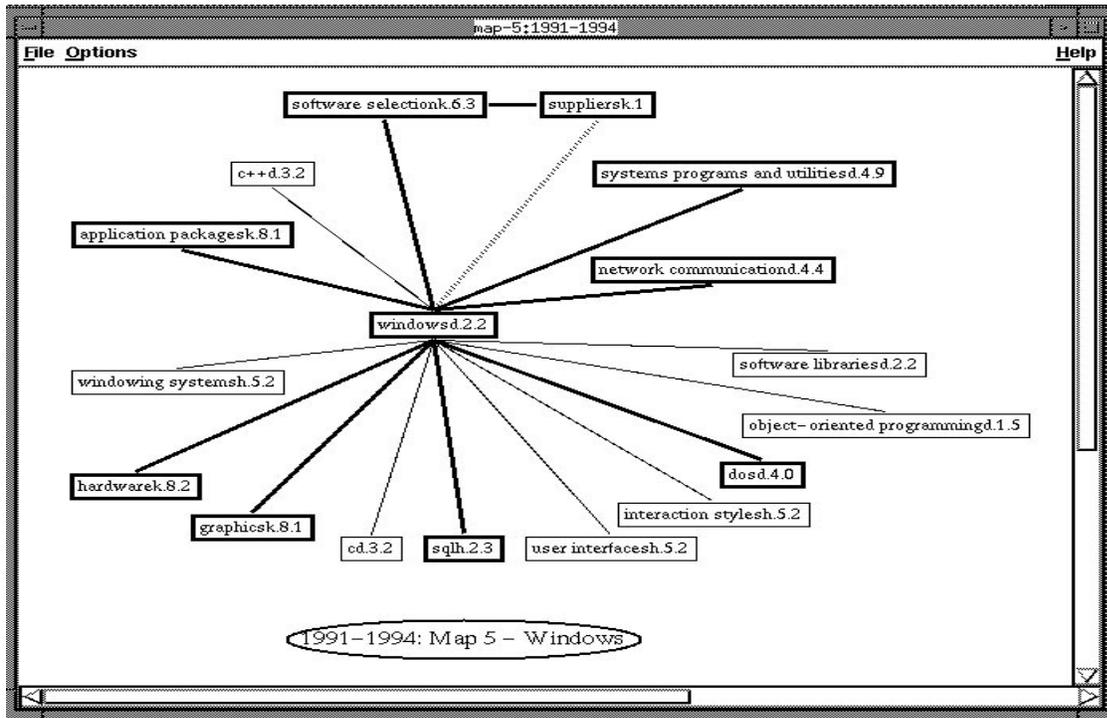


Figure A.3-5: Windows

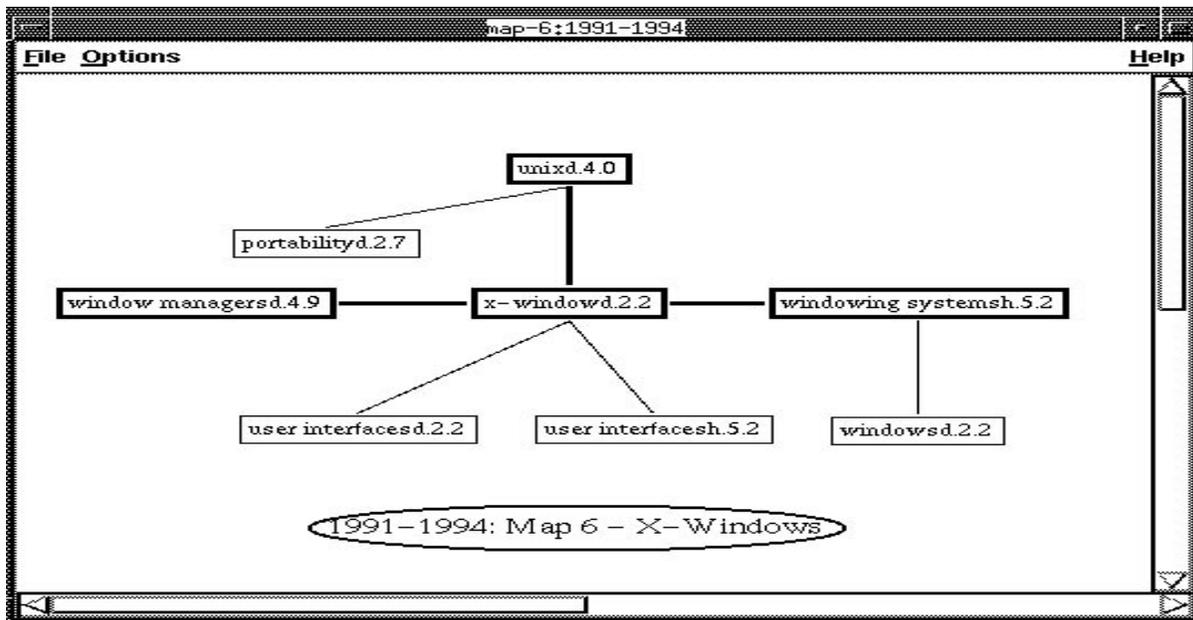


Figure A.3-6: X-Windows

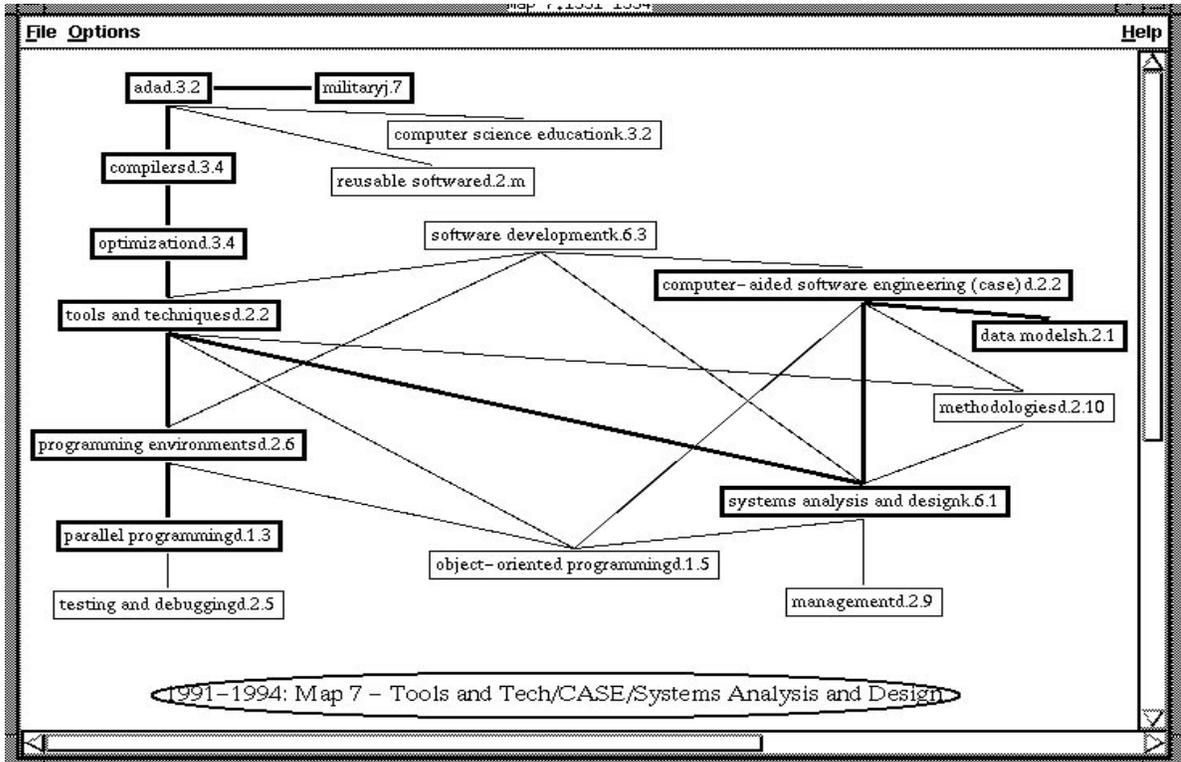


Figure A.3-7: Tools and Techniques - CASE - Systems Analysis and Design

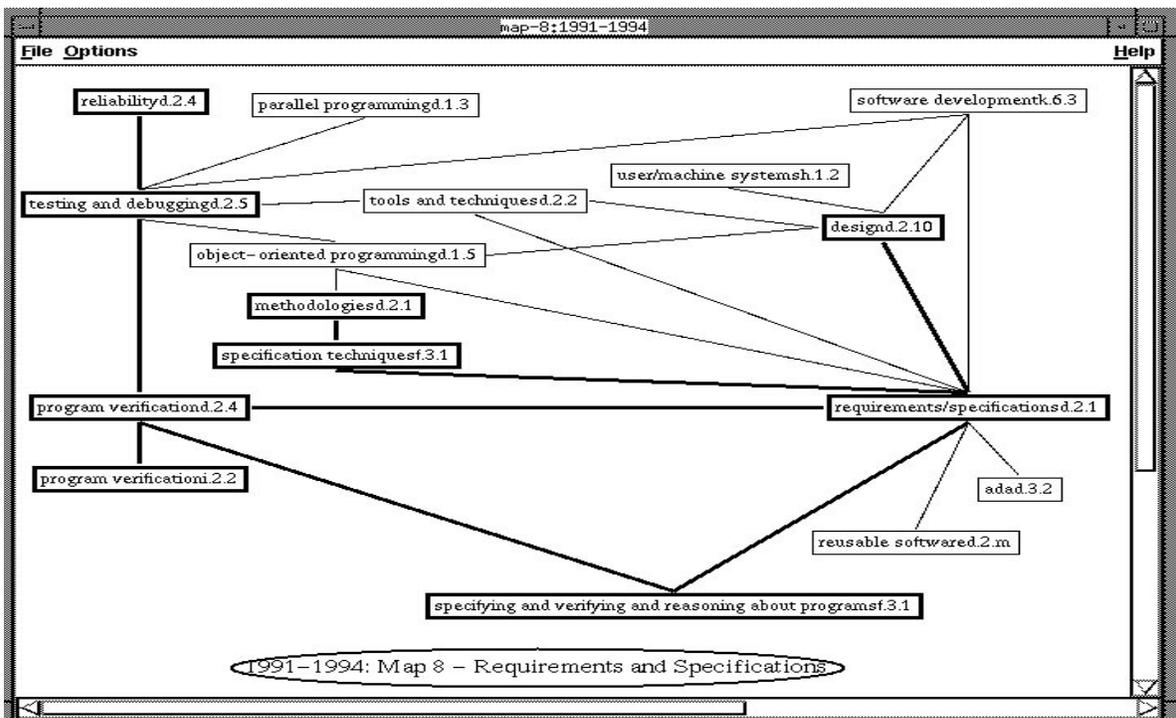


Figure A.3-8: Requirements/Specifications

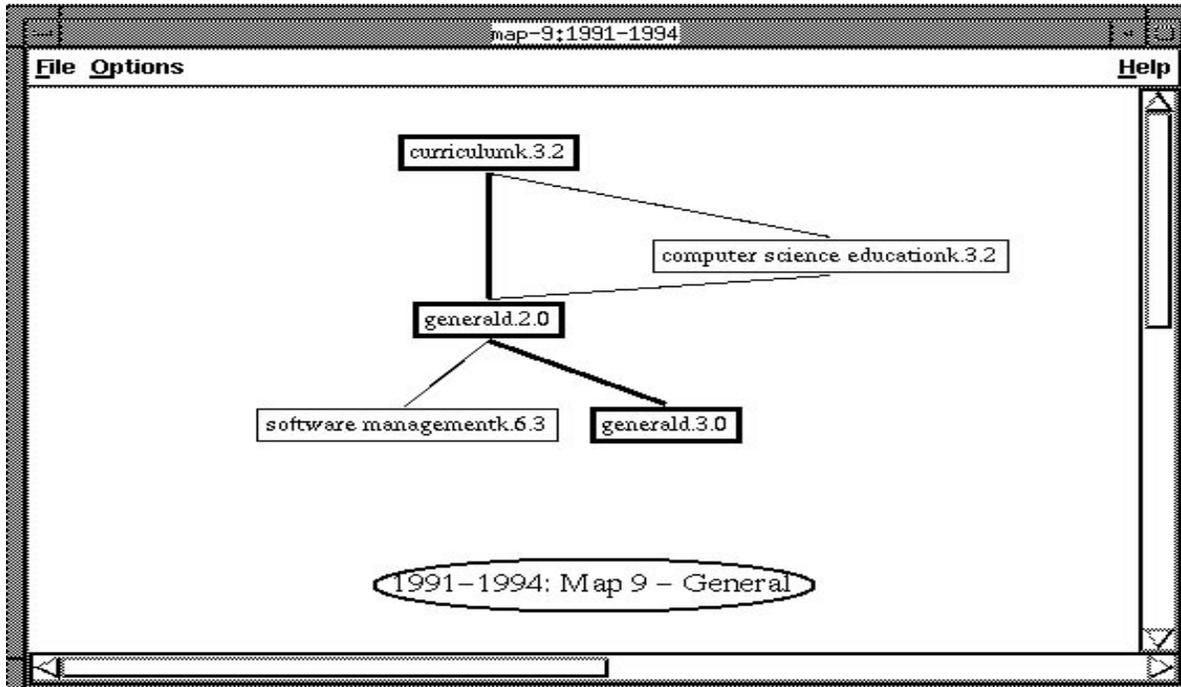


Figure A.3-9: General

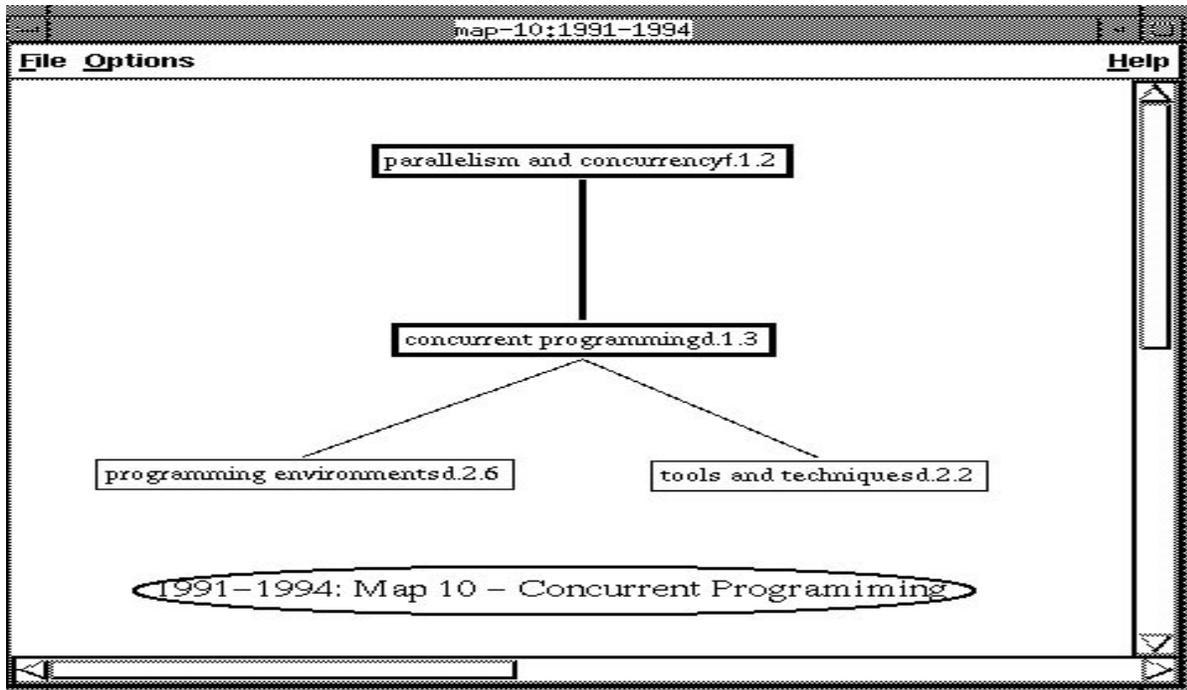


Figure A.3-10: Concurrent Programming

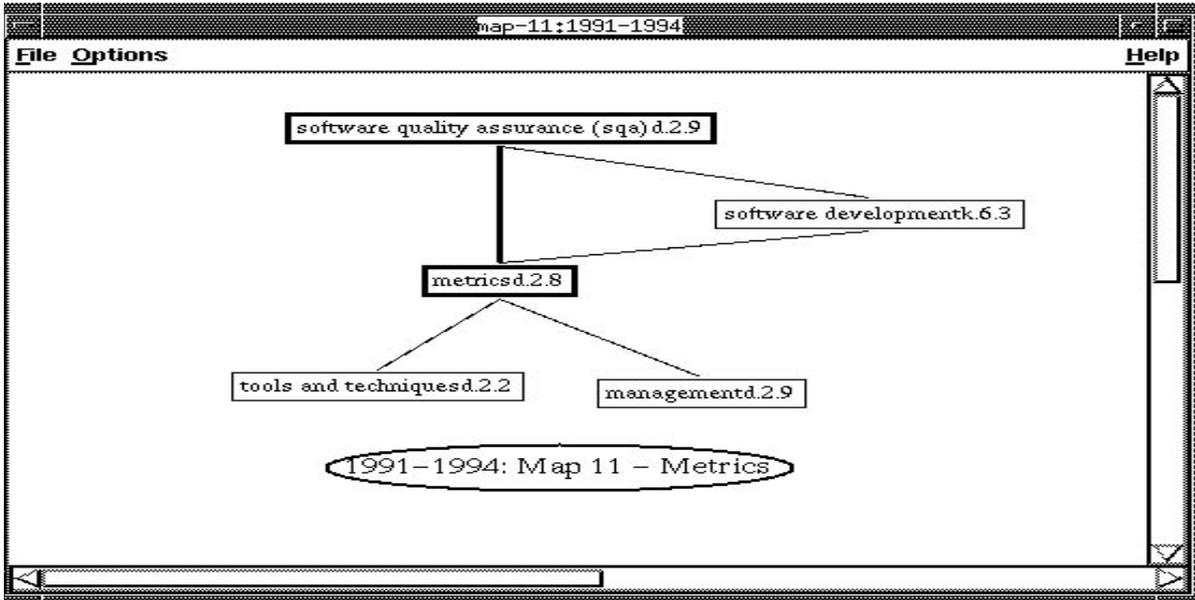


Figure A.3-11: Metrics