Reverse Engineering and System Renovation — An Annotated Bibliography —

M.G.J. van den Brand[1] P. Klint[1,2] C. Verhoef[1]

[1] Programming Research Group, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
[2] Department of Software Technology
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
email: markvdb@fwi.uva.nl, paulk@cwi.nl, chris@fwi.uva.nl

Abstract

To facilitate research in the field of reverse engineering and system renovation we have compiled an annotated bibliography. We put the contributions not only in alphabetical order but also grouped by topic so that readers focusing on a certain topic can read their annotations in the alphabetical listing. We also compiled an annotated list of pointers to information about reverse engineering and system renovation that can be reached via Internet. For the sake of ease we also incorporated a brief introduction to the field of reverse engineering.

Key Words & Phrases: Reverse engineering, Annotated bibliography, System renovation
1991 CR Categories: A.2, D.2.2, D.2.7, D.2.m, K.6.3
note: The authors were all in part sponsored by bank ABN AMRO, software house DPFinance, and the Dutch Ministery of Economical Affairs via the Senter Project #ITU95017 "SOS Resolver". The last author was also supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organization for Scientific Research (NWO), project Interactive tools for program understanding, *612-33-002.*

1 Executive Summary

There is a constant need for updating and renovating business-critical software systems for many and divers reasons: business requirements change, technological infrastructure is modernized, the government changes laws, or the third millennium approaches, to mention a few. Therefore, that in the area of software engineering the subjects of reverse engineering and system renovation become more and more important. The interest in such subjects originates from the difficulties that one encounters when attempting to maintain extremely large software systems. Such software systems are often called legacy systems, since it is a legacy of many different people that have developed and maintained them. It is not hard to understand that it is very difficult—if not impossible—to maintain them.

To make the problems a bit more concrete we will compare such software renovation projects to the renovation of a house. The problem that software engineers encounter could very well be stated in house renovation terms as the query: "How to renovate your house with the additional constraint that you want to use all the facilities of it during this renovation?" For many business-critical systems the same situation holds: how to renovate your software system while at the same time business continues as usual. An often heard (naïve) solution is to throw away the software as soon as a totally new system is finished (this is sometimes called shadowing). In house renovation terminology this would mean that you would have to build a completely new house and when this is finished you have to move the furniture from the old house to the new house before you can start using the new one. Then you can tear down the old one. It will be clear that this will be too expensive and that the shipping of the furniture will take too much time to meet the additional constraint. In software renovation terms the option of building a totally new system and throwing away the old one is for the same reasons as with a house renovation project usually too expensive and often even impossible since the shipping of the "furniture" (say, databases) from the old system to the new system will take weeks. So a more sophisticated renovation strategy seems necessary.

Before the actual renovation can start it will be necessary to make an inventory of the specification and the documentation of the system to be renovated. Also at this point there is a challenge for software engineers since the old systems lack mostly these sources of information. It is our experience that either there is no documentation at all, or the original programmers that could possibly explain the functionality of parts of the system have left, or both. The only documentation that is left is the source code itself. Thus, since the vital information of the software is solely accessible via the source code it will be necessary to develop tools to facilitate the renovation—a task for software engineers.

We hope to have elucidated that there is a need for sophisticated analysis of software to be used in renovation methodologies for large software systems, and that research on this issue is useful and important. A step towards a sound analysis of software renovation research is to study and analyze the literature on this subject, hence, this annotated bibliography.

We want to stress that the bibliography is intended to be useful for people who want to know more about reverse engineering and system renovation. More precicely, many of the entries in our bibliography are of a techical nature so researchers, practicioners and students interested in reverse engineering will hopefully benefit from them. We also included pointers to management issues and legal sides of reverse engineering and system renovation meant for people interested in those aspects.

It is not complete but instead gives pointers for further reading.

1.1 Related work

In [Arnold, 1993] another annotated bibliography can be found. There are also some on-line bibliographies. We discuss them in section 4.

1.2 Organization of the paper

In section 2 we give a brief impression of terminology in reverse engineering where we follow [Chikofsky and Cross, 1990]. In the next section (section 3) we sort the selected references by topic. In section 4 we give pointers to other sources of information: we provide a list of so-called universal resource locators. We give a short description of what can be expected when connecting to them. Finally, in section 5 we provide the annotated bibliography.

Acknowledgements We thank Arie van Deursen for discussions and comments on an earlier version of this paper.

2 Reverse engineering and system renovation in a nutshell

The term reverse engineering finds its origins in hardware technology and denotes the process of obtaining the specification of complex hardware systems. Now the meaning of this notion has shifted to software. As far as we know there is not (yet) a standard definition of what reverse engineering is but in [Chikofsky and Cross, 1990] we can read:

> "Reverse engineering is the process of analyzing a subject system to identify the system's components and their inter-relationships, and to create representations of the system in another form at higher levels of abstraction."

According to [Chikofsky and Cross, 1990] the following six terms characterize system renovation:

- forward engineering,
- reverse engineering,

- redocumentation,
- design recovery,
- restructuring,
- reengineering (or renovation).

Forward engineering moves from a high-level abstraction and design to a low-level implementation. *Reverse engineering* can be seen as the inverse process. It can be characterized as analysing a software system in order to, firstly, identify the system components and their interactions, and to, secondly, make representations of the system on a different, possible higher, level of abstraction.

Reverse engineering restricts itself to *investigating* a system. Adaptation of a system is beyond reverse engineering but within the scope of system renovation. *Redocumentation* focuses on making a semantically equivalent description at the same level of abstraction. It is in fact a simple form of reverse engineering. Tools for redocumentation include, among others, pretty printers, diagram generators, and cross-reference listing generators. In *design recovery* domain knowledge and external information is used to make an equivalent description of a system at a higher level of abstraction. So, more information than the source code of the system is used. The notion *restructuring* amounts to transforming a system from one representation to another one at the same level of abstraction. An essential aspect of restructuring is that the semantic behaviour of the original system and the new one should remain the same; no modifications of the functionality is involved. The purpose of *reengineering* or *renovation* is to study the system, by making a specification at a higher abstraction level, adding new functionality to this specification and develop a completely new system on the basis of the original one by using forward engineering techniques.

3 Classification

Papers addressing reverse engineering and system recovery can be classified in various categories. The classification that we propose is based on the material that we found. First, we list some introductory contributions and mention conferences dedicated to reverse engineering. Then we will proceed with program understanding and design recovery, reusability, maintainability, and program slicing. Then we list contributions that deal with the reverse engineering of more specific topics: data and data bases, user interfaces, and reverse engineering for a number of languages. Then we list formal techniques, tools and their implementation issues, restructuring and regeneration, testing, management, and miscellaneous contributions.

Introduction and general topics Articles that give an introduction to the field of reverse engineering and that define the relevant notions are [Biggerstaff, 1989; Byrne, 1992; Chikofsky and Cross, 1990; Osborne and Chikofsky, 1990]. Books that put the subject in context and contain a lot of introductory material are: [Arnold, 1993; Brodie and Stonebraker, 1995; Lano and Haughton, 1994; Zuylen, 1993]. A book that is possibly interesting is [Takeshita, 1992], but since it is in Japanese we are not able to give more information. A recent overview of research questions is given in [Selfridge *et al.*, 1993]. A tutorial on reverse engineering is [Rekoff, 1985].

Several conferences and workshops exist in this field, such as *Conference on Software Maintenance* (e.g., [Kellner, 1992]) and *Working Conference on Reverse Engineering* (e.g., [Waters and Chikofsky, 1993]).

Program understanding and design recovery There are many recent papers on this subject. We provide an extensive list: [Bertels *et al.*, 1993; Rugaber and Clayton, 1993; Beck and Eichmann, 1993; Blazy and Facon, 1994; Biggerstaff *et al.*, 1993; Biggerstaff, 1989; Cross, 1993; Cohen, 1995; Canfora *et al.*, 1993b; Cordy *et al.*, 1990; Cutillo *et al.*, 1993; Choi and Scacchi, 1990; Chen *et al.*, 1990; Devanbu *et al.*, 1991; Edwards and Munro, 1993; Hall, 1995; Harandi and Ning, 1990; Howden and Pak, 1992; Hausler *et al.*, 1990; Johnson *et al.*, 1992; Ning *et al.*, 1993; Oman and Cook, 1990; Olshefski and

Cole, 1993; Ornburn and Rugaber, 1992; Paul and Prakash, 1994b; Quilici, 1993; Rugaber *et al.*, 1990; Ritsch and Sneed, 1993; Rich and Wills, 1990; Johnson and Soloway, 1985; Vliet, 1993; Wills, 1993; Waters and Chikofsky, 1993; Wilde *et al.*, 1992; Zuylen, 1993].

Reuse In these papers the focus is on how to prevent the situation of legacy systems, that is, make the software easier to maintain by programming in replaceable and reusable components. [Gomaa, 1993; Hall, 1992; Khajenoori *et al.*, 1994; Mii and Takeshita, 1993; Whittle and Ratcliffe, 1993],

Maintainability To give an impression of the status of maintainability as a research field we give 2 quotations. Schneidewind [Schneidewind, 1987a] wrote in 1987 in his introduction to a special section on software maintenance that this "subject has received relatively little attention from the research community." Gallagher and Lyle write in [Gallagher and Lyle, 1991] "While some may view software maintenance as a less intellectually demanding activity than development, the central premise of this work is that software maintenance is *more* demanding." Here are some pointers to the maintainability subject: [Basili, 1990; Bennet *et al.*, 1992; Freeman and Layzell, 1994; Gallagher and Lyle, 1991; Jørgensen, 1995; Schneidewind, 1987b; Kellner, 1992; Pickard and Carter, 1995; Paul and Prakash, 1994a; Hartmann and Robson, 1990],

Program slicing A survey of program slicing techniques is given in [Tip, 1995]. Other references are: [Beck and Eichmann, 1993; Ning *et al.*, 1993; Cutillo *et al.*, 1993; Gallagher and Lyle, 1991; Olshefski and Cole, 1993; Binkley, 1992; Hall, 1995; Weiser, 1984; Gupta *et al.*, 1992].

Reverse engineering of data and databases One the first books on data reverse engineering is [Aiken, 1995]. Other references are: [Aiken *et al.*, 1993; Premerlani and Blaha, 1993; Hainaut *et al.*, 1993; Ritsch and Sneed, 1993; Brodie and Stonebraker, 1995; Tangorra and Chiarolla, 1995; Tan and Ling, 1995].

Reverse engineering of user interfaces [Merlo *et al.*, 1993].

Reverse engineering of specific languages Many business-critical systems are written in COBOL. So a number of papers geared towards this language are available. For other languages there are also contributions.

COBOL [Lano and Haughton, 1993; Edwards and Munro, 1993; Edwards and Munro, 1993; Gray *et al.*, 1995; Ning *et al.*, 1993; Cutillo *et al.*, 1993; Hainaut *et al.*, 1993; Ritsch and Sneed, 1993; Newcomb and Markosian, 1993; Zuylen, 1993; Sneed, 1992]

Pascal [Canfora *et al.*, 1993b]

C [Chu and Patel, 1992; Olshefski and Cole, 1993; Wilde *et al.*, 1992],

Lisp [Wills, 1993].

Ada [Canfora *et al.*, 1993a; Cross, 1993; Gray *et al.*, 1995].

Fortran [Byrne, 1991]

CHILL [Vogt and Hendriks, 1994].

Formal techniques In the following list formal techniques are used in which CCS, Denotational semantics, Petri nets, Z, and Z++ are applied to approach certain problems in reverse engineering. [Bowen *et al.*, 1993; Baumann *et al.*, 1994; Baumann *et al.*, 1993; Horwitz *et al.*, 1989; Keller *et al.*, 1995; Lano and Haughton, 1993; Merlo *et al.*, 1993; Ward and Bennett, 1993; Zuylen, 1993; Bennet *et al.*, 1992]

Tools and implementation techniques Many tools have been implemented to aid in various reverse engineering tasks. For an elaborate hypertext page on CASE tool vendors we refer to section 4, where a pointer to an electronically available index is given. [Baumann *et al.*, 1994; Baumann *et al.*, 1993; Reubenstein *et al.*, 1993; Ning *et al.*, 1993; Canfora *et al.*, 1992; Canfora *et al.*, 1993b; Cutillo *et al.*, 1993; Keller *et al.*, 1995; Miller and Straus III, 1987; Mendelzon and Sametinger, 1995; Olshefski and Cole, 1993; Cross, 1993; Rich and Waters, 1990; Reubenstein *et al.*, 1993; Wills, 1993; Ward and Bennett, 1993; Wilde and Huitt, 1991; Newcomb and Markosian, 1993; Zuylen, 1993; Bennet *et al.*, 1992; Johnson *et al.*, 1992; Cordy *et al.*, 1990; Paul and Prakash, 1994a; McGinnes, 1994; Tilley *et al.*, 1994; Rich and Wills, 1990]

Restructuring, transformation, and regeneration In this category, methods and tools are described to perform actual reverse engineering tasks. [Bennet *et al.*, 1992; Chu and Patel, 1992; Kozaczynski *et al.*, 1992; Hutchens and Basili, 1985; Miller and Straus III, 1987; Newcomb and Markosian, 1993; Oulsnam, 1982; Pleszkoch *et al.*, 1992; Johnson and Soloway, 1985; Urschler, 1975; Waters, 1988; Williams and Ossher, 1978; Ward and Bennett, 1993; Ramshaw, 1988; Zuylen, 1993; Choi and Scacchi, 1990]

Testing After modification of software systems it is necessary to test the new system. Some pointers are [Kung *et al.*, 1993; Binkley, 1992; Laski and Szemer, 1992; Gupta *et al.*, 1992; Hartmann and Robson, 1990]

Management Many reverse engineering projects are huge. The management of such projects is not at all trivial and gets attention in: [Aiken *et al.*, 1993; Arnold, 1993; Ward and Bennett, 1993; Zuylen, 1993]

Miscellaneous Certain issues that are important, but do not fit our classification in a natural way are [Ammarguellat, 1992; Hecht, 1977; Marlowe and Ryder, 1990; Belady and Evangelisti, 1981; Ashcroft and Manna, 1972; Samuelson, 1990; Withrow, 1990]. We mention that the subjects go from go to elimination to legal aspects of reverse engineering.

4 Other sources of information

In [Arnold, 1993] another annotated bibliography can be found. Noteworthy, perhaps, is that this annotated bilbiography mainly contains other references than ours.

Nowadays, much information is not only available via books and journals but also via Internet. In this section we listed some universal resource locators (URLs) that are related to reverse engineering and system renovation, including some that contain an on-line bibliography. Of course, we have made ourselves a page that contains the URLs below. Contact `http://adam.fwi.uva.nl/~chris/reverse.html` for both a dvi file of this bibliography and an hypertext version of the list below.

- `http://www.cc.gatech.edu/reverse/` is a site where the Georgia tech reverse engineering group presents their papers, tools and members. Moreover, pointers to other groups are given.
- `http://www.erg.abdn.ac.uk/users/brant/sre/` is a site that gives information like who is who in reverse engineering. Hyperlinks to research institutes, universities, and other sites are grouped together in a number of interest areas. Furthermore, some introductory information is available. Researchers in the field are encouraged to contribute to this WWW site under construction.
- `http://www.year2000.com/cgi-bin/clock.cgi/` This is a site of a firm specialized in the year 2000 problem. It contains information on how to join a mailing list on this subject.
- `http://www.software.ibm.com/year2000/index.html` This is a URL also dedicated to the year 2000 problem. A comprehensive set of services, tools and support is available to help customers prepare for the Year 2000 transition.

- `ftp://lscftp.kgn.ibm.com/pub/year2000/y2kpaper.ps` is a URL from which a manual can be obtained dedicated to the year 2000 problem.

- `http://www.qucis.queensu.ca/Software-Engineering/vendor.html` is a URL to a CASE tool vendor index. This is useful for tools that can be used in reverse engineering.

- `http://www.csc.tntech.edu/~linos/` is a page describing research activities related to program comprehension and reengineering performed at the CARE (Computer-Aided Reengineering) Laboratory in the Computer Science Department of Tennessee Technological University.

- `http://stout.levtech.com/levtech-marketing/papers.html` this is a bibliography of reengineering papers based on software refinery.

- `http://www-cs.open.ac.uk/~jonrob/bibliog.html` this is an online bibliography on software and reuse. At the time of writing this paper contained 114 items.

- `http://www.scism.sbu.ac.uk/cios/islam/ReuseBib.txt` This URL is a bibliography of software reuse papers written after 1990.

- `http://www.informatik.uni-stuttgart.de/ifi/ps/reengineering/reengineering.html` is an on-line bibliography on reverse engineering. It contains also the abstracts of the papers.

- `http://128.172.188.1/isydept/faculty/paiken/drebib.htm` is a pointer to a reengineering bibliography that can be obtained on a floppy disk. For more details please contact the owner of the page.

5 Annotated Bibliography

[Aiken *et al.*, 1993] P. Aiken, A. Muntz, and R. Richards. A framework for reverse engineering DoD legacy information systems. In *[Waters and Chikofsky, 1993]*, pages 180–191, 1993. *Gives an overview of the reverse engineering methodology used inside the DoD for the reengineering of information systems.*

[Aiken, 1995] P. Aiken. *Data Reverse Engineering: Slaying the Legacy Dragon.* McGraw-Hill, 1995. *This is the first book describing the process of recovering data architectures from existing information systems and using it to develop a foundation for enterprise integration and other reengineering efforts.*

[Ammarguellat, 1992] Z. Ammarguellat. A control-flow normalization algorithm and its complexity. *IEEE Transactions on Software Engineering*, 18(3):237–251, 1992. *A simple method is presented for normalizing the control-flow of programs to facilitate program transformations, program analysis, and automatic parallelization. This method does not make use of code replication. The normalization results in a restructuring of the code that obviates the need for control dependency relations.*

[Arnold, 1993] R.S. Arnold. *Software Reengineering.* IEEE Computer Society Press, 1993. *In this book an introduction to software reengineering is provided. Context and definitions of key notions are included. Then various subjects are treated in the form of a collection of papers that are reprinted from other sources. Subjects that we can find are: business process reengineering, the connection with economics, experiences with real-life reengineering projects, evaluation of tools used in such projects, the technological aspects of reengineering, data reengineering and its migration problems, source code analysis, restructuring and translation, the annotation and documentation of existing programs, reusability aspects, design recovery, the object oriented approach to recovery, program understanding, and knowledge based program analysis. This book contains an annotated bibliography.*

[Ashcroft and Manna, 1972] E. Ashcroft and Z. Manna. The translation of goto programs into while programs. In C.V. Freiman, J.E. Griffith, and J.L. Rosenfeld, editors, *Proceedings of IFIP Congress 71*, volume 1, pages 250–255. North-Holland, 1972. *It is shown that every flowchart program can be written without go to statements by using while statements. The transformation does not give rise to less efficient programs and, moreover, the structure of the original flowchart program is preserved.*

[Basili, 1990] V. Basili. Viewing maintenance as reuse oriented software development. *IEEE Software*, 7(1):19–25, 1990. *In this paper the maintenance process is incorporated in the life-cycle perspective geared towards the reusability of the various components.*

[Baumann *et al.*, 1993] P. Baumann, J. Fässler, M. Kiser, and Z. Öztürk. Beauty and the Beast or A Formal Description of the Control Constructs of Cobol and its Implementation. Technical Report 93.39, Department of Computer Science, University of Zurich, Switzerland, 1993. *A formal semantics for a subset of COBOL is presented with the aid of denotational semantics. The subset consists of the control constructs of COBOL. In [Baumann* et al.*, 1994] it is argued that precisely this subset is relevant for reverse engineering.*

[Baumann *et al.*, 1994] P. Baumann, J. Fässler, M. Kiser, Z. Öztürk, and L. Richter. Semantics-based reverse engineering. Technical Report 94.08, Department of Computer Science, University of Zurich, Switzerland, 1994. *Denotational semantics is advocated as a formal foundation for program understanding. The ideas are implemented in a tool for reverse engineering called AEMES. This tool is geared towards COBOL-74.*

[Beck and Eichmann, 1993] J. Beck and D. Eichmann. Program and interface slicing for reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 54–63, 1993. *Describes the use of program slicing for the reverse engineering of Ada packages.*

[Belady and Evangelisti, 1981] L. Belady and C. Evangelisti. System partitioning and its measure. *Journal of Systems and Software*, 2:23–29, 1981. *A method to perform automatic clustering of data structures and calls is described. A metric to quantify the complexity of the resulting partitioning is given.*

[Bennet *et al.*, 1992] K. Bennet, T. Bull, and H. Yang. A transformation system for maintenance: turning theory into practice. In *[Kellner, 1992]*, pages 146–155, 1992. *Describes the architecture of the Maintainer's Assistant, a reverse engineering tool based on program transformations. Discusses the role of metrics in selecting appropriate transformations. Also see [Ward and Bennett, 1993].*

[Bertels *et al.*, 1993] K. Bertels, Ph. Vanneste, and C. de Backer. A cognitive approach to program understanding. In *[Waters and Chikofsky, 1993]*, pages 1–7, 1993. *Presents a method of program understanding based on a cognitive model of programming knowledge. The approach involves the generation of a high level, abstract, description that is robust with respect to conceptual errors and syntactic variations.*

[Biggerstaff *et al.*, 1993] T. Biggerstaff, B. Mitbander, and D. Webster. The concept assignment problem in program understanding. In *[Waters and Chikofsky, 1993]*, pages 27–43, 1993. *The problem of discovering abstract human oriented concepts and relating them to their implementation oriented counterparts is called the* concept assignment problem*. Describes various heuristic clues, as supported by the DESIRE system, that can be used for concept extraction.*

[Biggerstaff, 1989] T.J. Biggerstaff. Design recovery for maintenance and reuse. *IEEE Computer*, 22(7):36–49, 1989. *Design recovery uses the source code of a system as well as external information, such as documentation, personal experience, and knowledge of problem and application domain, to make a higher level abstraction. The key property is the formalization of informal information and domain knowledge.*

[Binkley, 1992] D. Binkley. Using semantic differencing to reduce the cost of regression testing. In *[Kellner, 1992]*, pages 41–50, 1992. *Gives an algorithm using dependence graphs and program slicing to partition a modified program in parts with affected program behaviour and parts with unaffected behaviour. Only the parts with affected behaviour have to be re-tested.*

[Blazy and Facon, 1994] S. Blazy and P. Facon. Partial evaluation for the understanding of FORTRAN programs. *International Journal of Software Engineering and Knowledge Engineering*, 4(4):535–559, 1994. *A technique and a tool are described supporting the partial evaluation of FORTRAN programs in order to understand old programs that have become very complex due to numerous alterations.*

[Bowen *et al.*, 1993] J. Bowen, P. Breuer, and K. Lano. A compendium of formal techniques for software maintenance. *Software Engineering Journal*, 8(5):253–262, 1993. *An overview of formal techniques developed recently to aid the software maintenance process and particularly reverse engineering is given.*

[Brodie and Stonebraker, 1995] M.L. Brodie and M. Stonebraker. *Migrating Legacy Systems — Gateways, Interfaces & The Incremental Approach*. Morgan Kaufmann Publishers, Inc., 1995. *This book gives a detailed description of strategies for migrating legacy systems. It advocates an incremental approach for the migration instead of doing it in* one *step. The legacy system is analyzed and the components to be updated are identified. The legacy system and the new system work in parallel and are connected via gateways. Migrated components are removed from the legacy system and added to the new system. The crucial steps in this process are establishing the right ordering of the components to be migrated and the use of powerful gateways. It is preferable not to develop these gateways yourself but to obtain them from third party software producers. A number of case-studies is presented and these case-studies demonstrate that these gateways are crucial even if all the code of the legacy system becomes obsolete. The book concludes with an extensive list of third party software producers which produce gateways.*

[Byrne, 1991] E.J. Byrne. Software reverse engineering: A case study. *Software—Practice and Experience*, 21(12):1349–1364, 1991. *Experience report describing the problem of reimplementing a Fortran program in Ada. Instead of a one-to-one translation, the original Fortran program is analyzed and design information is extracted which is then used to reimplement the program in Ada.*

[Byrne, 1992] E. Byrne. A conceptual foundation for software re-engineering. In *[Kellner, 1992]*, pages 226–235, 1992. *A conceptual foundation for software reengineering is presented yielding a general model of software reengineering. This model is described and is shown to be useful for examining reengineering issues such as the reengineering process and strategies for reengineering.*

[Canfora *et al.*, 1992] G. Canfora, A. Cimitile, and U. de Carlini. A logic-based approach to reverse engineering tools production. *IEEE Transactions on Software Engineering*, 18(12):1053–1064, 1992. *Difficulties arising during the use of documents produced by reverse engineering tools are discussed and analyzed.*

[Canfora *et al.*, 1993a] G. Canfora, A. Cimitile, and U. De Carlini. A reverse engineering process for design level document production from ada code. *Information and Software Technology*, 35(1):23–34, 1993. *A reverse engineering process for producing design level documents by static analysis of ADA code is described. This is achieved via concurrent data flow diagrams describing the task structure and the data flow between tasks.*

[Canfora *et al.*, 1993b] G. Canfora, A. Cimitile, and M. Munro. A reverse engineering method for identifying reusable abstract data types. In *[Waters and Chikofsky, 1993]*, pages 73–82, 1993. *Describes a methodology and experimental Prolog-based tool for the extraction of reusable data type declarations from source code. Illustrated for a medium-size Pascal program.*

[Chen *et al.*, 1990] Y-F. Chen, M.Y. Nishimoto, and C.V. Ramamoorthy. The C information abstraction system. *IEEE Transactions on Software Engineering*, 16(3):325–334, 1990. *A system for analyzing program structures is described. The applications of this system include: generation of graphical views, subsystem extraction, program layering, dead code elimination, and binding analysis.*

[Chikofsky and Cross, 1990] E.J. Chikofsky and J.H. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990. *Definitions of a number of key notions in the field of reverse engineering are proposed. Forward and reverse engineering, redocumentation, design recovery, restructuring, and reengineering are described.*

[Choi and Scacchi, 1990] S.C. Choi and W. Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, 7(1):66–71, 1990. *An algorithm is described that for a given initial design description the system-reconstruction algorithm constructs a hierarchy of the system's modules and subsystems.*

[Chu and Patel, 1992] W Chu and S. Patel. Software restructuring by enforcing localization and information hiding. In *[Kellner, 1992]*, pages 165–172, 1992. *Starting with information describing function calls and global variable usage, this paper presents a clustering technique that generates Ada-like packages describing the structure of a given software system. Has been applied to several existing systems implemented in C.*

[Cohen, 1995] W.W. Cohen. Inductive specification recovery: Understanding software by learning from example behaviors. *Automated Software Engineering*, 2:107–129, 1995. *A method for program understanding that does not rely on parse-and-recognize techniques (as advocated in, for example, [Rich and Waters, 1990]) is presented. After the code has been annotated the system is run on a number of representative test cases, generating from the annotations examples of the behaviour. Finally, inductive learning techniques are used to generalize the examples, thus forming an abstract, general description of the behaviour of the annotated code.*

[Cordy *et al.*, 1990] J.R. Cordy, N.L. Eliot, and M.G. Robertson. Turingtool: A user interface to aid in the software maintenance task. *IEEE Transactions on Software Engineering*, 16(3):294–301, 1990. *In this paper the approach of viewing a program in a structured way is advocated. With the aid of queries the user can influence the view of the program and can, therefore, get a better idea of what the program is doing. Things that are not important for a certain view are elided, but can be accessed by clicking on them—the elided text becomes visual. The program can also be edited with this tool.*

[Cross, 1993] J. Cross. Reverse engineering of control structure diagrams. In *[Waters and Chikofsky, 1993]*, pages 107–116, 1993. *Describes a tool for the automatic generation of a new graphical representation for Ada software (Control Structure Diagrams). These diagrams aim at improving the comprehension of Ada programs and can potentially replace the original source code.*

[Cutillo *et al.*, 1993] F. Cutillo, P. Fiore, and G. Visaggio. Identification and extraction of "domain independent" components in large programs. In *[Waters and Chikofsky, 1993]*, pages 83–92, 1993. *Uses program slicing to extract components from COBOL programs by means of Viasoft's tools INSIGHT, SMARTDOC and RENAISSANCE.*

[Devanbu *et al.*, 1991] P. Devanbu, R.J. Bachman, P.G. Selfridge, and B.W. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 34(5):35–49, 1991. *A system called LaSSIE (Large Software System Information Environment) is presented. It incorporates a large knowledge base, a semantic retrieval algorithm based on formal inference, and a powerful user interface incorporating a graphical browser and a natural language parser. The system is intended to help programmers find useful information about large software systems.*

[Edwards and Munro, 1993] H. Edwards and M. Munro. RECAST: reverse engineering from COBOL to SSADM specifications. In *[Waters and Chikofsky, 1993]*, pages 44–53, 1993. *Describes methodology and tooling for the extraction of SSADM diagrams from COBOL programs.*

[Freeman and Layzell, 1994] M.J. Freeman and P.J. Layzell. A meta-model of information systems to support reverse engineering. *Information and Software Technology*, 36(5):283–294, 1994. *A method is discussed to help software maintainers to gain a richer understanding of a software system and its components. This is achieved by enhancing traditional reverse-engineering tools and prevents the loss of knowledge during forward engineering.*

[Gallagher and Lyle, 1991] K. Gallagher and J. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, 1991. *In this paper the technique of program slicing is used to facilitate maintenance of software systems by extending the notion of program slice to a so-called decomposition slice (a slice that captures all computation on a given variable).*

[Gomaa, 1993] H. Gomaa. A reuse-oriented approach for structuring and configuring distributed applications. *Software Engineering Journal*, 8(2):61–71, 1993. *For the design of configurable distributed applications it is advocated to develop reusable specifications and architectures. Then targets can be generated by tailoring the reusable specifications and architectures. The method is elucidated by way of an example.*

[Gray *et al.*, 1995] R. Gray, T. Bickmore, and S. Williams. Reengineering cobol systems to ada. Technical report, InVision Software Reengineering, Software Technology Center, Lockheed Palo Alto Laboratories, 1995. *This paper describes the reengineering of 50,000 lines of Cobol code and the translation to Ada. The goal was to do it as automatically as possible. An inferential method was used to obtain all needed information from the Cobol code itself, no external information from users or programmers was needed. The authors claim that inferential methods will be the basis of the reengineering technology of the 21th century.*

[Gupta *et al.*, 1992] R. Gupta, M. Harrold, and M. Soffa. An approach to regression testing using slicing. In *[Kellner, 1992]*, pages 299–308, 1992. *A new approach to data flow based regression testing is described that uses program slicing algorithms to detect definition-use pairs that are affected by a program change. The advantage of this approach is that neither the data flow history nor a recomputation of data flow is necessary.*

[Hainaut *et al.*, 1993] J.-L. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 161–170, 1993. *Gives a methodology for recovering the conceptual schema of databases. Illustrated with various COBOL examples.*

[Hall, 1992] P.A.V. Hall. Overview of reverse engineering and reuse research. *Information and Software Technology*, 34(4):239–249, April 1992. *It is argued in this paper that reuse of steps taken in forward engineering—such as ideas, prototypes, temporary solutions, etc—should be stored somehow so that new systems do not need to be developed from scratch. This is indeed useful when a system that is developed while saving such information needs reverse engineering but for legacy systems this is too late.*

[Hall, 1995] R.J. Hall. Automatic extraction of executable program subsets by simultaneous dynamic program slicing. *Automated Software Engineering*, 2:33–53, 1995. *An algorithm to automatically extract a correctly functioning subset of the code of a system is presented. The technique is based on computing a simultaneous dynamic program slice of the code for a set of representative inputs. Experiments show that the algorithm produces significantly smaller subsets than with existing methods.*

[Harandi and Ning, 1990] M.T. Harandi and J.Q. Ning. Knowledge-based program analysis. *IEEE Software*, 7(1):74–81, 1990. *Automatic program analysis with a tool called PAT is used to understand programs on a high level. The applications are maintenance for large complex programs.*

[Hartmann and Robson, 1990] J. Hartmann and D.J. Robson. Techniques for selective revalidation. *IEEE Software*, 7(1):31–36, 1990. *A systematic and automated approach is discussed to effectively revalidate modified software while minimizing the time and cost involved in maintenance testing.*

[Hausler *et al.*, 1990] P.A. Hausler, M.G. Pleszkoch, R.C. Linger, and A.R. Hevner. Using function abstraction to understand program behavior. *IEEE Software*, 7(1):55–63, 1990. *In this paper it is avocated to improve the understanding of programs by structuring them. The authors think that the potential exists for an automated tool to take unstructured code and derive its functionality.*

[Jørgensen, 1995] M. Jørgensen. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering*, 21(8):674–681, 1995. *Eleven software maintenance effort prediction models are discussed.*

[Hecht, 1977] M. Hecht. *Flow analysis of computer programs.* Elsevier North-Holland, 1977. *A classical book on the theroy and implementation of algorithms for data flow analysis.*

[Horwitz *et al.*, 1989] S. Horwitz, T. Reps, and J. Prins. Integrating non-interfering versions of programs. *ACM Transactions on Programming Languages and Systems*, 11(3):345–387, 1989. *In this paper the program-integration problem is formalized. An algorithm is given that produces an intergrated program from two variations of a base program. The algorithm is semantics-based rather than text-based. The algorithm assumes a programming language containing only simple programming constructs, like assignment statements, conditional statements, and iterative statements.*

[Howden and Pak, 1992] W. Howden and S. Pak. Problem domain, structural and logical abstractions in reverse engineering. In *[Kellner, 1992]*, pages 214–224, 1992. *Introduces a formal notation for documenting various aspects of existing software. Has been applied, manually, to sample COBOL programs.*

[Hutchens and Basili, 1985] D. Hutchens and V. Basili. System structure analysis: clustering with data bindings. *IEEE Transactions on Software Engineering*, SE-11(8):749–757, 1985. *The use of cluster analysis as a tool for system modularization is examined. It appears that the clustering of data bindings provides a meaningful view of system modularization.*

[Johnson and Soloway, 1985] W. Johnson and E. Soloway. PROUST: knowledge-based program understanding. *IEEE Transactions on Software Engineering*, SE-11(3):267–275, 1985. *This paper describes a tool to help novice programmers to learn how to program. It is based on a knowledge base and has also a tutoring aspect. The tool is not intended for large scale program understanding but the ideas underlying this paper may very well be applicable to it.*

[Johnson *et al.*, 1992] B. Johnson, S. Ornburn, and S. Rugaber. A quick tools approach to program analysis and software maintenance. In *[Kellner, 1992]*, 1992. *Describes the use of standard Unix tools like (Awk, Lex, Yacc) for extracting information from PL/M code. The information is then visualized using a commercial CASE tool (Software Through Pictures).*

[Keller *et al.*, 1995] R.K. Keller, X. Shen, R. Lajoie, M. Ozkan, and T. Tao. Environment support for business reengineering: the Macrotec approach. *Software—Concepts and Tools*, 16(1):31–40, 1995. *A business reengineering approach is developed based on the formalism of coloured Petri-nets. The Macrotec environment has been engineered for the support and validation of this approach. Macrotec is based on Macronets, the latter being a variation of the Petri net formalism.*

[Kellner, 1992] M. Kellner, editor. *Proceedings Conference on Software Maintenance*. IEEE Computer Society Press, 1992. *Several papers in these proceedings that are directly related to reverse engineering are discussed separately in this bibliography.*

[Khajenoori *et al.*, 1994] S. Khajenoori, D.G. Linton, and C.A. Morris. Enhancing software reusability through effective use of the essential modelling approach. *Information and Software Technology*, 36(8):495–501, 1994. *It is advocated to develop new software systems by reusing design components from existing ones. With the aid of the so-called essential modelling approach it is possible to determine reusable components.*

[Kozaczynski *et al.*, 1992] W. Kozaczynski, J. Ning, and A. Engberts. Program concept recognition and transformation. *IEEE Transactions on Software Engineering*, 18(12):1065–1075, 1992. *An approach to automated concept recognition and its application to maintenance-related program transformations is described. An interesting point here is that transformation of code can be expressed as transformation of abstract concepts.*

[Kung *et al.*, 1993] C. Kung, JH. Gao, P. Hsia, J. Lin, and Y. Toyoshima. Design recovery for software testing of object-oriented programs. In *[Waters and Chikofsky, 1993]*, pages 202–211, 1993. *Describes a methodology for testing OO software.*

[Lano and Haughton, 1993] K. Lano and H. Haughton. Integrating formal and structured methods in reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 17–26, 1993. *Describes the integration of formal (Z++) and structured (SSADM) methods in reverse engineering as prototyped in the REDO project.*

[Lano and Haughton, 1994] K. Lano and H. Haughton. *Reverse Engineering and Software Maintenance — A Practical Approach*. McGraw-Hill, 1994. *This book describes a fundamental approach to reverse engineering and software maintenance. After an introduction in software maintenance and reverse engineering a number of tools and approaches are discussed to tackle various problems in these areas. An elaborate introduction in logic and program semantics is given. One method (the process model) to address maintenance and reverse engineering is discussed in more detail. The book concludes with a number of case-studies which use a formal approach based on logic and program semantics.*

[Laski and Szemer, 1992] J. Laski and W. Szemer. Identification of program modifications and its applications in software maintenance. In *[Kellner, 1992]*, pages 282–290, 1992. *One of the problems in software maintenance is the revalidation of modified code. Such a process should preferably be restricted only to those parts of the program that are affected by the modifications. In this paper a formal method is described to identify modifications made in a program.*

[Marlowe and Ryder, 1990] Marlowe and Ryder. Properties of data flow frameworks. A unified model. *Acta Informatica*, 28:121–163, 1990. *An overview of data flow frameworks and their characterizing properties is given. Contains many references to the field of data flow analysis.*

[McGinnes, 1994] S. McGinnes. CASE support for collaborative modelling: re-engineering conceptual modelling techniques to exploit the potential of CASE tools. *Software Engineering Journal*, 9(4):183–189, 1994. *It is advocated that more benefit would be obtained if both analysis and design techniques were reengineered so as to make the best possible use of CASE tools. Ways on how to achieve this are given in the paper using examples from a prototype CASE tool.*

[Mendelzon and Sametinger, 1995] A. Mendelzon and J. Sametinger. Reverse engineering by visualizing and querying. *Software—Concepts and Tools*, 16(4):170–182, 1995. *A tool called Hy+ is described that can be used for reverse engineering. Hy+ is a general-purpose data visualization system for querying and visualizing information about object-oriented software systems. Hy+ supports this for arbitrary graph-like databases. The use is demonstrated with the evaluation of software metrics, verifying constraints and identifying design patterns.*

[Merlo *et al.*, 1993] E. Merlo, J. Girard, K. Kontogiannis, P. Panangaden, and R. De Mori. Reverse engineering of user interfaces. In *[Waters and Chikofsky, 1993]*, pages 171–179, 1993. *Extracts user interface descriptions from COBOL/CICS source code and translates them to abstract behaviour descriptions based on process algebra (CCS). Part of the analysis is done using Refine/COBOL. The translation itself is done manually.*

[Mii and Takeshita, 1993] N. Mii and T. Takeshita. Software re-engineering and reuse from a Japanese point of view. *Information and Software Technology*, 35(1):45–53, 1993. *The use of the concept of reusable pieces of software as parts in the Japanese situation is reviewed. It is more geared towards preventive forward engineering than to reverse engineering.*

[Miller and Straus III, 1987] J. Miller and B. Straus III. Implications of automatic restructuring cobol. *ACM Sigplan Notices*, 22(6):76–82, 1987. *The question whether or not mechanical transformations of unstructured program code to a structured equivalent can provide an improvement in the understanding of that program is addressed. As an example the language COBOL is examined. The paper also discusses a tool (called Structured Retrofit) that performs such transformations for COBOL mechanically.*

[Newcomb and Markosian, 1993] Ph. Newcomb and L. Markosian. Automating the modularization of large COBOL programs: application of an enabling technology for reengineering. In *[Waters and Chikofsky, 1993]*, pages 222–230, 1993. *Experience report using the Software Refinery to build a modularization tool for COBOL.*

[Ning *et al.*, 1993] J. Ning, A. Engberts, and W. Kozaczynski. Recovering reusable components from legacy systems. In *[Waters and Chikofsky, 1993]*, pages 64–72, 1993. *Gives an overview of the program segmentation facilities of the COBOL/SRE system, which are based on various forms of program slicing.*

[Olshefski and Cole, 1993] D. Olshefski and A. Cole. A prototype system for static and dynamic program understanding. In *[Waters and Chikofsky, 1993]*, pages 93–106, 1993. *Describes the experimental PUNDIT system that combines static and dynamic information for program understanding. It comprises a static analyzer for C source code and a, mostly language-independent, graphical user interface. Gives various examples of program views.*

[Oman and Cook, 1990] P.W. Oman and C.R. Cook. The book paradigm for improved maintenance. *IEEE Software*, 7(1):39–45, 1990. *It is shown that traditional typographical formats used in books work very well to aid program understanding.*

[Ornburn and Rugaber, 1992] S. Ornburn and S. Rugaber. Reverse engineering: resolving conflicts between expected and actual software designs. In *[Kellner, 1992]*, pages 32–40, 1992. *Experience report describing the application of the Synchronized Refinement method [Rugaber* et al.*, 1990] to a real-time embedded system.*

[Osborne and Chikofsky, 1990] W. Osborne and E. Chikofsky, editors. *Special issue on Maintenance, reverse engineering and design recovery.* IEEE Software, 7(1):11–105. 1990. *In this special issue a number of papers dealing with various aspects of reverse engineering are collected. Most of the individual papers are discussed in this annotated bibliography.*

[Oulsnam, 1982] G. Oulsnam. Unraveling unstructured programs. *The Computer Journal*, 25(3):379–387, 1982. *A method for transforming unstructured program flowcharts into structured ones is presented. The form of the derived structured programs is such that the original unstructured programs can be easily recovered, thus revealing what overheads in space and time are inherent in the structured forms.*

[Paul and Prakash, 1994a] S. Paul and A. Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994. *It is argued that existing solutions to locating source code fragments that match certain patterns are insufficient. A framework in which pattern languages are used to specify interesting code features is presented. These are obtained by extending the source programming language with pattern-matching symbols. This is implemented in a tool called SCRUPLE.*

[Paul and Prakash, 1994b] S. Paul and A. Prakash. Supporting queries on source code: A formal framework. *International Journal of Software Engineering and Knowledge Engineering*, 4(3):325–348, 1994. *A source code query system is a powerful mechanism to obtain crucial information necessary to successfully performing a reverse engineering task. A source code algebra (SCA) is developed which is strongly based on relational algebras as well as on many sorted algebras. Two types of data types are distinguished in the source code algebra model:*

- *atomic data types, such as integer, float, etc.*
- *composite data types (so-called objects):*
  - *singular objects, such as while-statement, identifier, etc.*
  - *collective objects, such as statement-list, etc.*

The objects are extended with four kinds of attributes, namely, components, references, annotations, and methods. An extensive set of source code algebra operators are defined, such operators defined for atomic data types, individual objects, and collections, i.e., sets and sequences. The operators for the collections are strongly influenced by the operators from the relational algebra domain.

[Pickard and Carter, 1995] M.M. Pickard and B.D. Carter. A field study of the relationship of information flow and maintainability of cobol programs. *Information and Software Technology*, 37(4):195–202, 1995. *The results of a field study of the relationship of information flow to the maintainability of COBOL modules in a data processing environment are presented. There is a significant correlation between maintainability and information flow and with (information flow) metrics it is possible to identify poorly maintained modules.*

[Pleszkoch *et al.*, 1992] M. Pleszkoch, R. Linger, and A. Hevner. Eliminating non-traversable paths from structured programs. In *[Kellner, 1992]*, pages 156–164, 1992. *Considers the problem of control variables (i.e., ranging over the Booleans or some small enumeration type) that obscure the structure of otherwise structured programs. Control flow is represented by regular expressions which are further processed (subset construction) to find a version of the program without redundant control paths.*

[Premerlani and Blaha, 1993] W. Premerlani and M. Blaha. An approach for reverse engineering of relational databases. In *[Waters and Chikofsky, 1993]*, pages 151–160, 1993. *Experience report describing the reverse engineering of several relational databases to OMT (Object Modeling Technique) diagrams. The process is partly automated using a variety of tools.*

[Quilici, 1993] A. Quilici. A hybrid approach to recognizing programming plans. In *[Waters and Chikofsky, 1993]*, pages 126–133, 1993. *Based on an experiment regarding human understanding of a given C program, a new organization for a plan library is presented. It consists of a plan definition, a plan recognition rule, and specialized constraints. Extends the plan library developed by Andersen Consulting.*

[Ramshaw, 1988] L. Ramshaw. Eliminating goto's while preserving program structure. *Journal of the ACM*, 35(4):893–920, 1988. *A method is described to eliminate GO-TO statements from a program while the program's original structure is being preserved.*

[Rekoff, 1985] M. Rekoff. On reverse engineering. *IEEE Transactions on Systems, Man and Cybernetics*, 3/4:244–252, 1985. *This paper is a tutorial on reverse engineering that defines some key notions.*

[Reubenstein *et al.*, 1993] H. Reubenstein, R. Piazza, and S. Roberts. Separating parsing and analysis in reverse engineering tools. In *[Waters and Chikofsky, 1993]*, pages 117–125, 1993. *Experience report describing the extension of an existing analysis tool with a new syntactic front-end. Concludes that language-independence as well separation of parsing and analysis are essential for extensibility.*

[Rich and Waters, 1990] C. Rich and R.C. Waters. *The Programmer's Apprentice*. Addison-Wesley, 1990. *This book, named after the project it reports on, is intended both to serve as an example of a general method to the builders of many and diverse computer-aided design tools and to study how software is analyzed, modified, verified, and documented with the goal to automate such typically software engineering tasks. A demonstration system has been completed within the Programmer's Apprentice project that illustrates most of the key capabilities of it, albeit that this system is restricted to the task of program implementation.*

[Rich and Wills, 1990] C. Rich and L.M. Wills. Recognizing a program's design: A graph-parsing approach. *IEEE Software*, 7(1):82–89, 1990. *In this paper it is assumed that most programmers use similar structures to program. Such so-called cliches can be recognized automatically and can then be used to generate the documentation of the program.*

[Ritsch and Sneed, 1993] H. Ritsch and H. Sneed. Reverse engineering programs via dynamic analysis. In *[Waters and Chikofsky, 1993]*, pages 192–201, 1993. *Describes a dynamic analysis of COBOL programs. By inspection of transaction files assertions are generated capturing the input and output requirements of each database operation.*

[Rugaber and Clayton, 1993] S. Rugaber and R. Clayton. The representation problem in reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 8–16, 1993. *Choosing the proper representation to build models describing software entities during reverse engineering is the* representation problem. *This paper examines the representation problem by presenting a taxonomy of models and representations.*

[Rugaber *et al.*, 1990] S. Rugaber, S.B. Ornburn, and R.J. LeBlanc. Recognizing design decisions in programs. *IEEE Software*, 7(1):46–54, 1990. *In this paper it is advocated that in order to effectively maintain an existing system, the maintenance programmer must be able to sustain decisions made earlier in the design process. To accomplish this, she/he must be able to recognize and understand this decisions. A way is given to characterize such decisions.*

[Samuelson, 1990] P. Samuelson. Reverse-engineering someone else's software: is it legal? *IEEE Software*, 7(1):90–96, 1990. *The legal issues concerning reverse engineering are discussed: does reverse engineering software infringe intellectual-property law?.*

[Schneidewind, 1987a] N. Schneidewind. Introduction to the special section on software maintenance. *IEEE Transactions on Software Engineering*, SE-13(3):301, 1987. *This preface introduces a special section on software maintenance.*

[Schneidewind, 1987b] N. Schneidewind. The state of software maintenance. *IEEE Transactions on Software Engineering*, SE-13(3):303–310, 1987. *An overview of the state of the art in software maintenance and criticizes the apparent disinterest in the research field is provided.*

[Selfridge *et al.*, 1993] P. Selfridge, R. Waters, and E. Chikofsky. Challenges for the field of reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 144–150, 1993. *This position paper presents ten challenges for improvement of reverse engineering research in three areas: (a) avoiding artificial data; (b) focusing on concrete economic and technical impact; and (c) facilitating researcher communication by establishing standard terminology and selecting standard data sets.*

[Sneed, 1992] H. Sneed. Migration of procedurally oriented COBOL programs in an object-oriented architecture. In *[Kellner, 1992]*, pages 105–116, 1992. *The subject of this paper is to describe the migration of procedurally structured COBOL into functionally equivalent object-oriented programs. Their major differences are described together with an approach to bridge the gap between the two.*

[Takeshita, 1992] T. Takeshita. *Software Maintenance/Re-engineering and Reuse*. Kyoritsu Shuppan, Tokyo, 1992. (In Japanese).

[Tan and Ling, 1995] H.B.T. Tan and T.W. Ling. Recovery of object-oriented design from existing data-intensive business programs. *Information and Software Technology*, 37(2):67–77, 1995. *A method is given for the recovery of a specification from an existing data-intensive business program using an augmented model that is proposed in the paper.*

[Tangorra and Chiarolla, 1995] F. Tangorra and D. Chiarolla. A methodology for reverse engineering hierarchical databases. *Information and Software Technology*, 37(4):225–231, 1995. *The steps of a reverse engineering process for translating a hierarchical data scheme into a conceptual description in the extended entity-relationship model are described. Contains a case study.*

[Tilley *et al.*, 1994] S.R. Tilley, K. Wong, M-A.D. Storey, and H.A. Müller. Programmable reverse engineering. *International Journal of Software Engineering and Knowledge Engineering*, 4(4):501–520, 1994. *This paper argues that most reverse engineering environments are not flexible enough. They are directed towards the tool builders instead of the users of the environments. Besides a number of basic facilities, such as parsing, the reverse engineering tool should allow a high level of extensibility. The authors present an existing scripting language, Tcl, to enable users to develop their own routines for graph layout, metrics and analysis. Most generic reverse engineering environments break down if they have to deal with millions of lines of code. The constructed abstract syntax trees contain too much information. The reverse engineering environment should allow a flexible gathering of information, not only based on abstract syntax trees. The way the information is gathered should be programmable. The reverse engineering environments should be reusable in various application domains. The user of the environment should be able to program the the environment to make it suited for a specific application domain.*

[Tip, 1995] F. Tip. A survey of program slicing techniques. *Journal of programming languages*, 3:121–189, 1995. *Surveys the state-of-the-art in program slicing and gives many references to the literature.*

[Urschler, 1975] G. Urschler. The automatic restructuring of programs. *IBM Journal of Research and Development*, 19:181–194, 1975. *A method is described that allows the translation of an unstructured program into a set of top-down structured, semantically founded, go-to-free modules. This method leads to a certain amount of code replication.*

[Vliet, 1993] J.C. van Vliet. Automatische design recovery: een illusie? *Informatie*, 35(6):384–389, 1993. *(In Dutch.) The definition of reverse engineering by Chikofski and Cross [Chikofsky and Cross, 1990] is used to explain some reverse engineering terminology. The author demonstrates by some examples that domain specific knowledge is essential for successful design recovery. It is therefore essential that tools for design recovery contain a model of the application domain, in which concepts of the underlying domain with their relations and dependencies are modeled. It is not possible to have automatic design recovery because the concepts of the application domain can only be described by means of informal semantics.*

[Vogt and Hendriks, 1994] H.H. Vogt and P.R.H. Hendriks. Code-analyse in de praktijk. *Informatie*, 36(12):764–770, 1994. *(In Dutch.) Describes the RECALL-project aiming at the analysis of the complex software of telephone exchanges to identify the components and the interaction between these components. The RECALL reverse engineering prototype tool consists of a code browser and a CHILL parser, and it offers the following functionality: (a) holophrasting; (b) call graph view; (c) call sequence view; (d) all calls of a procedure view; and (e) jump history.*

[Ward and Bennett, 1993] M. Ward and K. Bennett. A practical program transformation system for reverse engineering. In *[Waters and Chikofsky, 1993]*, pages 212–221, 1993. *Uses program transformation techniques as a basis for reverse engineering. Source files are first translated into WSL (wide-spectrum language). By means of a large collection of WSL transformation and user guidance, the WSL program is simplified. Next it can be translated back into the original source language or into the specification language Z. The process is supported by the ReForm tool which contains parsers for IBM assembly language and for a Basic subset. Also see [Bennet* et al.*, 1992].*

[Waters and Chikofsky, 1993] R.C. Waters and E.J. Chikofsky, editors. *Proceedings of Working Conference on Reverse Engineering*. IEEE Computer Society Press, 1993. *All papers in these proceedings are discussed separately in this bibliography.*

[Waters, 1988] R.C. Waters. Program translation via abstraction and reimplementation. *IEEE Transactions on Software Engineering*, 14(8), 1988. *The translation paradigm of abstraction and reimplementation, which is one of the goals of the Programmer's Apprentice project [Rich and Waters, 1990] is presented. A translator has been constructed which translates Cobol programs into Hibol (a very high level, business data processing language).*

[Weiser, 1984] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984. *In this paper some properties of slices are presented. It is shown that the use of data-flow analysis is sufficient to find approximate slices of the generally unsolvable problem of finding statement-minimal slices.*

[Whittle and Ratcliffe, 1993] B. Whittle and M. Ratcliffe. Software component interface description for reuse. *Software Engineering Journal*, 8(6):307–318, 1993. *The development of a language CIDER, which stands for Component Interface Descriptor is described. It is an object-oriented language in which it is feasible to integrate and reuse component interfaces based on a model of the reusable software component.*

[Wilde and Huitt, 1991] N. Wilde and R. Huitt. A reusable toolset for software dependency analysis. *Journal of Systems and Software*, 14(2):97–102, 1991. *A general purpose tool set that has been developed to capture and analyse software dependencies is described. A prototype of this so-called dependency analysis tool set has been implemented to analyze C code.*

[Wilde *et al.*, 1992] N. Wilde, J. Gomez, T. Gust, and D. Strasburg. Locating user functionality in old code. In *[Kellner, 1992]*, pages 200–205, 1992. *Proposes a probabilistic technique to match expected functionality with the actual functions as implemented in existing code. An experiment reveals that the method works reasonable but cannot replace human experts.*

[Williams and Ossher, 1978] M. Williams and H. Ossher. Conversion of unstructured flow diagrams into structured form. *The Computer Journal*, 21(2):161–167, 1978. *Various already proposed methods to convert unstructured flow diagrams into equivalent structured ones are discussed. Moreover a general method for performing such conversions is discussed.*

[Wills, 1993] L. Wills. Flexible control for program recognition. In *[Waters and Chikofsky, 1993]*, pages 134–143, 1993. *Uses chart parsing (a graph-based parsing technique) for recognizing program plans. The GRASPR tool implements this technique and can be applied to Common Lisp programs (less than 1000 lines).*

[Withrow, 1990] C. Withrow. Error density and size in Ada software. *IEEE Software*, 7(1):26–30, 1990. *In this paper we can find an empirical study of the relation between error density and the length of an Ada module. The results show that there is an optimal length and that shorter modules and larger ones contain more errors. For reverse engineering such metrics can give an indication for the status of the software.*

[Zuylen, 1993] H. van Zuylen, editor. *The ReDo compendium: reverse engineering for software maintenance.* Wiley, 1993. *Gives an overview of the results of the REDO project and covers most aspects of reverse engineering. Various approaches are discussed: (a) compilation of COBOL programs to equational specifications, restructuring and simplification of these specifications, and regeneration of COBOL code from them; (b) compilation of COBOL to UNIFORM, an intermediate language supporting all features of both COBOL and JCL; (c) compilation of COBOL to COBOL-IF, a simplified syntactic representation of COBOL programs; (d) abstraction of the meaning of COBOL code in the form of Z++ specifications. Various experimental tools providing partial support for the above techniques are discussed. The results described in this book should be considered as useful experiments. Since the techniques have not been applied to a number of large scale projects the method does not yet constitute a mature reverse engineering methodology.*