

CHAPTER 10

SOFTWARE ENGINEERING TOOLS AND METHODS

David Carrington

Department of Computer Science and Electrical Engineering
The University of Queensland
Brisbane, Qld 4072 Australia
+61 7 3365 3310
davec@csee.uq.edu.au

Table of Contents

1	Introduction	1
2	Definition of the Software Engineering Tools and Methods Knowledge Area	1
3	Breakdown of Topics for Software Engineering Tools and Methods	1
4	Breakdown Rationale	5
5	Matrix of Topics vs. Reference Material.....	6
6	Recommended References for Software Engineering tools and Methods	7
Appendix A – References Used to Write and Justify the Knowledge Area Description		9

1 INTRODUCTION

This chapter provides an initial breakdown of topics within the Software Engineering Infrastructure Knowledge Area as defined by the document “Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge”. Earlier versions of this Knowledge Area included material on integration and reuse, but this has been removed. Consequently the Knowledge Area has been renamed from “Software Engineering Infrastructure” to “Software Engineering Tools and Methods”.

The five general software engineering texts [DT97, Moo98, Pfl98, Pre97, and Som96] have been supplemented as primary sources by “The Computer Science and Engineering Handbook” [Tuc96], which provides nine chapters on software engineering topics. Chapter 112, “Software Tools and Environments” by Steven Reiss [Rei96] is particularly helpful for this Knowledge Area. Additional specialized references are identified for particular topics.

One observation from assembling the guide to this knowledge area is that there is a scarcity of recent technical writing on practical software engineering tools. Obviously, there are detailed manuals on specific tools and numerous research papers on innovative software tools, but there is a

gap between the two. One difficulty is the high rate of change in software tools. Specific details alter regularly, making it difficult to provide up-to-date concrete examples. There also seems to be an attitude that software engineering tools are prosaic and not worthy of study beyond the level required for use.

2 DEFINITION OF THE SOFTWARE ENGINEERING TOOLS AND METHODS KNOWLEDGE AREA

The Software Engineering Tools and Methods Knowledge Area includes both the software development environments and the development methods knowledge areas identified in the Straw Man version of the guide.

Software development environments are the computer-based tools that are intended to assist the software development process. Tools allow repetitive, well-defined actions to be automated, thus reducing the cognitive load on the software engineer. The engineer is then free to concentrate on the creative aspects of the process. Tools are often designed to support particular methods, reducing any administrative load associated with applying the method manually. Like methods, they are intended to make development more systematic, and they vary in scope from supporting individual tasks to encompassing the complete life cycle.

Development methods impose structure on the software development activity with the goal of making the activity systematic and ultimately more likely to be successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks and guidelines for checking both the process and the product. Development methods vary widely in scope, from a single life cycle phase to the complete life cycle. The emphasis in this Knowledge Area is on methods that encompass multiple lifecycle phases since phase-specific methods are likely to be covered in other Knowledge Areas.

3 BREAKDOWN OF TOPICS FOR SOFTWARE ENGINEERING TOOLS AND METHODS

This section contains a breakdown of topics in the Software Engineering Tools and Methods Knowledge Area, with

brief descriptions and references. The Knowledge Area is partitioned at the top level into Software Tools and Software Methods. Two levels of references are provided with topics: the recommended references within brackets and additional references within parentheses. References to a particular chapter are denoted as *RefcN* where *N* is the chapter number. A similar denotation is used for references to a particular section *Ref:sN*. Figure 1 provides a diagrammatic representation of the breakdown of topics.

I. Software Tools

The partitioning of the Software Tools section uses the same structure as the Stone Man Version of the Guide to the Software Engineering Body of Knowledge. The first five subsections correspond to the five Knowledge Areas (Requirements, Design, Construction, Testing, and Maintenance) that correspond to a phase of a software lifecycle, so these sections provide a location for phase-specific tools. The next four subsections correspond to the remaining Knowledge Areas (Process, Quality, Configuration Management and Management), and provide locations for phase-independent tools that are associated with activities described in these Knowledge Areas. Two additional subsections are provided: one for infrastructure support tools that do not fit in any of the earlier sections, and a Miscellaneous subsection for topics, such as tool integration techniques, that are potentially applicable to all classes of tools. Because software engineering tools evolve rapidly and continuously, the hierarchy and description avoids discussing particular tools as far as possible.

A. Software Requirements Tools

Tools for dealing with software requirements have been partitioned into two topics: modeling and traceability. More fine-grained partitioning would certainly be possible but this partition was considered adequate based on the coverage of tools in the literature.

Requirements modeling tools

Tools used for eliciting, recording, analyzing and validating software requirements belong in this section.

Traceability tools

[Pre97:s29.3, DT97:s4.1, DT97:s12.3]

Requirements traceability tools are becoming increasingly important as the complexity of software systems grow, and since traceability tools are relevant also in other lifecycle phases, they have been separated from the other tools for requirements.

B. Software Design Tools

[]

This section covers tools for creating and checking software designs. There is a variety of such tools, with much of this variety being a consequence of the diversity of design notations and methods. While this variety of tools exists, no compelling partitions for this topic were found.

C. Software Construction Tools

Software construction tools are concerned with the production and translation of the program representation (commonly known as source code) that is sufficiently detailed and explicit to enable machine execution.

Program editors

Program editors are tools used for creation and modification of programs (and possibly associated documents). These tools can be general-purpose text or document editors, or they can be specialized for a target language. Editing refers to human-controlled development tools.

Compilers and code generators

Traditionally, compilers have been non-interactive translators of source code but there has been a trend to integrate compilers and program editors to provide integrated programming environments. This topic also covers pre-processors, linker/loaders, and code generators.

Interpreters

Interpreters provide software execution through emulation. They can support software construction activities by providing a more controllable and observable environment for program execution.

Debuggers

Debugging tools have been made a separate topic since they support the construction process but are different from program editors or compilers.

D. Software Testing Tools

Testing tools are categorized according to where in the testing process they are used.

Test generators

Test generators assist the development of test cases.

Test execution frameworks

Test execution frameworks enable the execution of test cases in a controlled environment where the behavior of the object under test is observed.

Test evaluation tools

Test evaluation tools support the assessment of the results of test execution, helping to determine whether the observed behavior conforms to the expected behavior.

Test management tools

Test management tools provide support for managing all aspects of the testing process.

Performance analysis tools []

This topic covers tools for measuring and analyzing software performance. It is a specialized form of testing where the goal is to assess the performance behavior rather than the functional behavior (correctness).

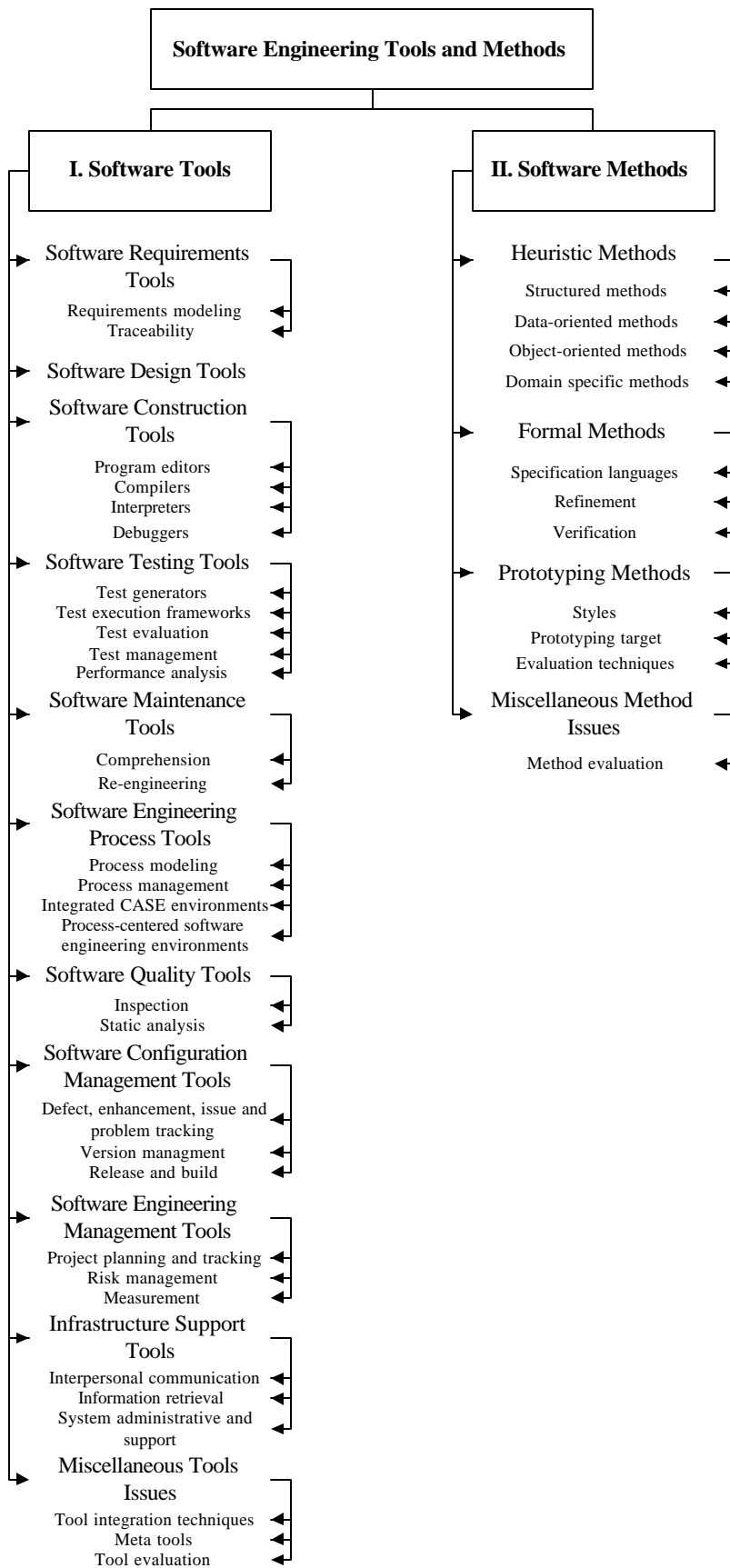


Figure 1 – Breakdown of topics in the software tools and methods knowledge area

E. Software Maintenance Tools

Software maintenance is often presented as additional iterations of the development lifecycle and consequently makes use of tools for all other phases. This category encompasses tools that have particular importance in software maintenance where an existing system is being modified. Two categories are identified: comprehension tools and re-engineering tools.

Comprehension tools

This topic concerns tools to assist human comprehension of programs. Examples include visualization tools such as animators and program slicers.

Re-engineering tools

Re-engineering tools allow translation of a program to a new programming language, or a database to a new format. Reverse engineering tools assist the process by working backwards from an existing product to create abstract artifacts such as design and specification descriptions, which then can be transformed to generate a new product from an old one.

F. Software Engineering Process Tools

Process modeling tools

This topic covers tools to model and investigate software processes.

Process management tools

Integrated CASE environments

(ECMA93, ECMA94, IEEE-1209, IEEE-1348, MNS96)

Computer-aided software engineering tools or environments that cover multiple phases of the software development lifecycle belong in this section. Such tools perform multiple functions and hence potentially interact with the software process that is being enacted.

Process-centered software engineering environments

(GJ96)

This topic covers those environments that explicitly incorporate software process information and that guide and monitor the user according to a defined process.

G. Software Quality Tools

Inspection tools

This topic covers tools to support reviews and inspections.

Static analysis tools

This topic deals with tools that analyze software artifacts, such as syntactic and semantic analyzers, and data, control flow and dependency analyzers. Such tools are intended for checking software artifacts for conformance or for verifying desired properties.

H. Software Configuration Management Tools

Tools for configuration management have been categorized as related to tracking issues associated with a particular software product, management of multiple versions of a

product or to managing the task of software release and build.

Defect, enhancement, issue and problem tracking tools

Version management tools

Release and build tools

This category includes installation tools that have become widely used for configuring the installation of software products.

I. Software Engineering Management Tools

Management tools are subdivided into three categories: project planning and tracking, risk management, and measurement.

Project planning and tracking tools

Risk management tools

Measurement tools

J. Infrastructure support tools

This section covers tools that provide interpersonal communication, information retrieval, and system administration and support. These tools, such as e-mail, databases, web browsers and file backup tools, are generally not specific to a particular lifecycle stage, nor to a particular development method.

Interpersonal communication tools

Information retrieval tools

System administration and support tools

K. Miscellaneous tool issues

This section covers issues that are applicable to all classes of tools. Three categories are identified: tool integration techniques, meta-tools and tool evaluation.

Tool integration techniques

[Som96:s25.2]

(Bro94)

Tool integration is important for making individual tools cooperate. This category potentially overlaps with integrated software engineering environments where integration techniques are applied, but it was felt that this topic is sufficiently distinct to merit its own category. The typical kinds of tool integration are platform, presentation, process, data, and control.

Meta tools

Meta-tools generate other tools; compiler-compilers are the classic example.

Tool evaluation

(IEEE-1209, IEEE-1348, Mos92, VB97)

Because of the continuous evolution of software engineering tools, tool evaluation is an essential topic.

II. Software Development Methods

The software development section is divided into four subsections: *heuristic methods* dealing with informal approaches, *formal methods* dealing with mathematically based approaches, *prototyping methods* dealing with software development approaches based on various forms of prototyping, and *miscellaneous method issues*. The first three subsections are not disjoint; rather they represent distinct concerns. For example, an object-oriented method may incorporate formal techniques and rely on prototyping for verification and validation. Like software engineering tools, methodologies evolve continuously. Consequently, the Knowledge Area description avoids naming particular methodologies as far as possible.

A. Heuristic methods

This subsection contains four categories: *structured*, *data-oriented*, *object-oriented* and *domain-specific*. The domain-specific category includes specialized methods for developing systems that involve real-time, safety or security aspects.

Structured methods

Data-oriented methods

Object-oriented methods

Domain-specific methods

B. Formal methods

This subsection deals with mathematically based development methods and is subdivided by different aspects of formal methods. The first topic is the specification notation or language used. Specification languages are commonly classified as model-oriented, property-oriented or behavior-oriented. The second topic deals with how the method refines (or transforms) the specification into a form that is closer to the desired final form of an executable program. The third topic covers the verification properties that are specific to the formal approach and covers both theorem proving and model checking.

Specification languages & notations

Refinement

Verification/proving properties

C. Prototyping methods

This subsection covers methods involving software prototyping and is subdivided into prototyping styles, targets and evaluation techniques.

Styles

(PB92:c1)

The topic of prototyping styles identifies the different approaches: throwaway, evolutionary and the executable specification.

Prototyping target

(PB92:c2)

Example targets of a prototyping method may be requirements, architectural design or the user interface.

Evaluation techniques

This topic covers how the results of a prototype exercise are used.

D. Miscellaneous method issues

The final subsection is intended to cover topics not covered elsewhere in the software method area. The only topic identified so far is method evaluation.

1. Method evaluation

4 BREAKDOWN RATIONALE

The Stone Man Version of the Guide to the Software Engineering Body of Knowledge conforms at least partially with the partitioning of the software life cycle in the ISO/IEC 12207 Standard [ISO95]. Some Knowledge Areas, such as this one, are intended to cover knowledge that applies to multiple phases of the life cycle. One approach to partitioning topics in this Knowledge Area would be to use the software life cycle phases. For example, software methods and tools could be classified according to the phase with which they are associated. This approach was not seen as effective. If software engineering tools and methods could be cleanly partitioned by lifecycle phase, it would suggest that this Knowledge Area could be eliminated by allocating each part to the corresponding life cycle Knowledge Area, e.g., tools and methods for software design to the Software Design Knowledge Area. Such an approach would fail to identify the commonality of, and interrelationships between, both methods and tools in different life cycle phases. However since tools are a common theme to most Knowledge Areas, several reviewers of Version 0.5 of this Knowledge Area suggested that a breakdown based on Knowledge Area for tools would be helpful. The Industry Advisory Board endorsed this suggestion.

There are many links between methods and tools, and one possible structure would seek to exploit these links. However because the relationship is not a simple “one-to-one” mapping, this structure has not been used to organize topics in this Knowledge Area. This means that these links are not always explicitly identified.

Some topics in this Knowledge Area do not have corresponding reference materials identified in the matrices in Appendix 2. There are two possible conclusions: either the topic area is not relevant to this Knowledge Area, or additional reference material needs to be identified. Feedback from reviewers will be helpful to resolve this issue.

5 MATRIX OF TOPICS VS. REFERENCE MATERIAL

I. Software Tools	CW96	DT97	Pf198	Pre97	Rei96	Som96	Was96
A. Software Requirements Tools		4.1 12.3		11.4.2, 29.3		26.2	
Requirements modeling tools							
Traceability tools		7.4					
B. Software Design Tools		12.3		29.3		26.2	
C. Software Construction Tools		12.3		29.3	112.2	26.1	
Program editors							
Compilers and code generators							
Interpreters							
Debuggers							
D. Software Testing Tools		12.3	7.7, 8.7	29.3	112.3	26.3	
Test generators							
Test execution frameworks							
Test evaluation tools							
Test management tools							
Performance analysis tools					112.5		
E. Software Maintenance Tools		12.3	10.5	29.3			
Comprehension tools					112.5		
Re-engineering tools							
F. Software Engineering Process Tools		12.3				25, 26, 27	
Process modeling tools			2.3, 2.4				
Process management tools							
Integrated CASE environments				29	112.3, 112.4		
Process-centered software engineering environments				29.6	112.5		
G. Software Quality Tools		12.3					
Inspection tools							
Static analysis tools	X		7.7	29.3	112.5	24.3	
H. Software Configuration Management Tools		12.3	10.5		112.3		
Defect, enhancement, issue and problem tracking tools				29.3			
Version management tools				29			
I. Software Engineering Management Tools		12.3					
Project planning and tracking tools				29.3			
Risk management tools							
J. Infrastructure Support Tools		12.3					
Interpersonal communication tools				29.3			
Information retrieval tools				29.3			
System administration and support tools				29.3			
K. Miscellaneous Tool Issues		12.3					
Tool integration techniques			1.8		112.4		X
Meta tools							
Tool evaluation			8.10				

II. Development Methods		CW96	DT97	Pfl98	Pre98	Som96	Was96	CW96
A. Heuristic Methods							X	
1.	Structured methods		4.2, 5.2	4.5	10-18	15		
2.	Data-oriented methods		4.2, 5.2		12.8			
3.	Object-oriented methods		5.1, 5.2	4.4, 7.5	19-23	6.3, 14		
4.	Domain-specific methods				15	16		
B. Formal Methods			5.4		24, 25	9-11, 24.4		
1.	Specification languages	X		4.5	24.4			
2.	Refinement				25.3			
3.	Verification/proving properties	X		5.7, 7.3		24.2		
C. Prototyping Methods					2.5	8	X	
1.	Styles		12.2	4.6, 5.6	11.4			
2.	Prototyping targets		12.2					
3.	Evaluation techniques							
D. Miscellaneous Method Issues								
1.	Method evaluation							

6 RECOMMENDED REFERENCES FOR SOFTWARE ENGINEERING TOOLS AND METHODS

This section briefly describes each of the recommended references.

[CW96] Edmund M. Clarke et al. Formal Methods: State of the Art and Future Directions.

This tutorial on formal methods explains techniques for formal specification, model checking and theorem proving, and describes some successful case studies and tools.

[DT97] Merlin Dorfman and Richard H. Thayer (eds.). Software Engineering, IEEE Computer Society Press.

This tutorial volume contains a collection of papers organized into chapters. The following papers are referenced (section numbers have been added to reference individual papers more conveniently in the matrices in the Appendix):

Chapter 4: Software Requirements Engineering and Software Design

4.1 Software Requirements: A Tutorial, Stuart Faulk

4.2 Software Design: An Introduction, David Budgen

Chapter 5: Software Development Methodologies

5.1 Object-oriented Development, Linda M. Northrup

5.2 Object-oriented Systems Development: Survey of Structured Methods, A.G. Sutcliffe

5.4 A Review of Formal Methods, Robert Vienneau

Chapter 7: Software Validation, Verification and Testing

7.4 Traceability, James D. Palmer

Chapter 12 Software Technology

12.2 Prototyping: Alternate Systems Development Methodology, J.M. Carey

12.3 A Classification of CASE Technology, Alfonso Fuggetta

[Pfl98] S.L. Pfleeger. Software Engineering — Theory and Practice, Prentice-Hall.

This text is structured according to the phases of a life cycle so that discussion of methods and tools is distributed throughout the book.

[Pre97] R.S. Pressman. Software Engineering — A Practitioner's Approach (4th Ed.), McGraw-Hill

Chapter 29 covers "Computer-Aided Software Engineering" including a taxonomy of case tools (29.3). There is not much detail about any particular class of tool but it does illustrate the wide range of software engineering tools. The strength of this book is its description of methods with chapters 10-23 covering heuristic methods, chapters 24 and 25 covering formal methods. Section 11.4 describes prototyping methods and tools.

[Rei96] Steven P. Reiss. Software Tools and Environments in The Computer Science and Engineering Handbook. CRC Press, 1996 .

This chapter from [Tuc96] provides an overview of software tools. The emphasis is on programming tools rather than tools for analysis and design although CASE tools are mentioned briefly.

[Som96] Ian Sommerville. Software Engineering (5th Ed.), Addison-Wesley.

Chapters 25, 26 and 27 introduce computer-aided software engineering with the emphasis being on tool integration and large-scale environments. Static analysis tools are covered in Section 24.3. Chapter 9, 10 and 11 introduce formal methods with formal verification being described in Section 24.2 and the Cleanroom method in Section 24.4. Prototyping is discussed in Chapter 8.

[Was96] Anthony I. Wasserman. Toward a Discipline of Software Engineering, IEEE Software, vol. 13, no. 6 Nov. 1996, pp. 23-31.

This general article discusses the role of both methods and tools in software engineering. Although brief, the paper integrates the major themes of the discipline.

APPENDIX A – REFERENCES USED TO WRITE AND JUSTIFY THE KNOWLEDGE AREA DESCRIPTION

- [Ber92] Edward V. Berard. Essays on Object-Oriented Software Engineering. Prentice-Hall, 1993.
- [BP92] W. Bischofberger and G Pomberger. Prototyping-oriented Software Development: Concepts and Tools. Springer-Verlag, 1992.
- [Bro94] Alan W. Brown et al. Principles of CASE Tool Integration. Oxford University Press, 1994.
- [CB95] D.J. Carney and A.W. Brown. On the Necessary Conditions for the Composition of Integrated Software Engineering Environments. In Advances in Computers, Volume 41, pages 157-189. Academic Press, 1995.
- [CW96] Edmund M. Clarke, Jeanette M. Wing et al. Formal Methods: State of the Art and Future Directions. ACM Computer Surveys, 28(4):626-643, 1996.
- [Col94] Derek Coleman et al. Object-Oriented Development: The Fusion Method. Prentice Hall, 1994.
- [CGR95] Dan Gaigen, Susan Gerhart and Ted Ralston. Formal Methods Reality Check: Industrial Usage, IEEE Transactions on Software Engineering, 21(2):90-98, February 1995.
- [DT97] Merlin Dorfman and Richard H. Thayer, Editors. Software Engineering. IEEE Computer Society, 1997.
- [ECMA93] ECMA. TR/55 Reference Model for Frameworks of Software Engineering Environments, 3rd edition, June 1993.
- [ECMA94] ECMA TR/69 Reference Model for Project Support Environments, December 1994.
- [Fin00] Anthony Finkelstein, Editor. The Future of Software Engineering. ACM, 2000.
- [GJ96] Pankaj K. Garg and Mehdi Jazayeri. Process-Centered Software Engineering Environments, IEEE Computer Society, 1996.
- [HOT00] William Harrison, Harold Ossher and Peri Tarr. Software Engineering Tools and Environments: A Roadmap. In [Fin00], pp. 263-277, 2000.
- [IEEE-1175] IEEE. Trial-Use Standard Reference Model for Computing System Tool Interconnections, IEEE Std 1175-1992.
- [IEEE-1209] IEEE. Recommended Practice for the Evaluation and Selection of CASE Tools, IEEE Std 1209-1992 (ISO/IEC 14102, 1995).
- [IEEE-1348] IEEE Recommended Practice for the Adoption of CASE Tools, IEEE Std 1348-1995 (ISO/IEC 14471).
- [ISO-12207] ISO/IEC Standard for Information Technology —Software Life Cycle Processes, ISO/IEC 12207 (IEEE/EIA 12207.0-1996), 1995.
- [JH98] Stan Jarzabek and Riri Huang. The Case for User-Centered CASE Tools, Communications of the ACM, 41(8):93-99, August 1998.
- [KPP95] B. Kitchenham, L. Pickard, and S.L. Pfleeger. Case Studies for Method and Tool Evaluation, IEEE Software, 12(4):52-62, July 1995.
- [Lam00] Axel van Lamsweerde. Formal Specification: A Roadmap. In [fin00], pp. 149-159, 2000.
- [Mey97] Bertrand Meyer. Object-oriented Software Construction (2nd Ed.). Prentice Hall, 1997.
- [Mul00] Hausi Müller et al. Reverse Engineering: A Roadmap. In [Fin00], pp. 49-60, 2000.
- [Moo98] James W. Moore. Software Engineering Standards: A User's Road Map. IEEE Computer Society, 1998.
- [Mos92] Vicky Mosley. How to Assess Tools Efficiently and Quantitatively, IEEE Software, 9(3):29-32, May 1992.
- (MNS96) H.A. Muller, R.J. Norman and J. Slonim (eds.). Computer Aided Software Engineering, Kluwer, 1996. (A special issue of Automated Software Engineering, 3(3/4), 1996).
- [PB96] Gustav Pomberger and Günther Blaschek. Object-orientation and Prototyping in Software Engineering. Prentice Hall, 1996.
- [Pfl98] Shari Lawrence Pfleeger. Software Engineering: Theory and Practice. Prentice Hall, 1998.
- [Pos96] R.M. Poston. Automating specification-based Software Testing. IEEE, 1996.
- [Pre97] Roger S. Pressman. Software Engineering: A Practitioner's Approach. 4th edition, McGraw-Hill, 1997.
- [Rei96] Steven P. Reiss. Software Tools and Environments, Ch. 112, pages 2419-2439. In Tucker [Tuc96], 1996.
- [RW92] C. Rich and R.C. Waters. Knowledge Intensive Software Engineering Tools, IEEE Transactions on Knowledge and Data Engineering, 4(5):424-430, October 1992.
- [Som96] Ian Sommerville. Software Engineering. 5th edition, Addison-Wesley, 1996.
- [SO92] Xiping Song and Leon J. Osterweil. Towards Objective, Systematic Design-Method Comparisons, IEEE Software, 9(3):43-53, May 1992.
- [Tuc96] Allen B. Tucker, Jr., Editor-in-chief. The Computer Science and Engineering Handbook. CRC Press, 1996.
- [VB97] Laura A. Valaer and Robert C. Babb II. Choosing a User Interface Development Tool. IEEE Software, 14(4):29-39, 1997
- [Vin90] Walter G. Vincenti. What Engineers Know and How They Know It: Analytical Studies from Aeronautical History. John Hopkins University Press, 1990.
- [Was96] Anthony I. Wasserman. Toward a Discipline of Software Engineering, IEEE Software, 13(6): 23-31, November 1996.
- [Wie98] Roel Wieringa. A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. ACM Computing Surveys, 30(4):459-527, 1998.