PUC

Series: Monografias em Ciência da Computação

Nº 7/83

TOWARDS A MODEL FOR EVALUATING SPECIFICATIONS

A. von Staa

A.R.C. da Rocha

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453

RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Series : Monografias em Ciência da Computação, Nº 7/83

Editor : Antonio L. Furtado May, 1983

TOWARDS A MODEL FOR EVALUATING SPECIFICATIONS*

A. von Staa

and

A.R.C. da Rocha

* Research supported by FINEP contract 3.2.82.0179.00

ABSTRACT

Software quality depends directly on the quality of its specification. In other words, in spite of all our efforts, it will be almost impossible to construct high quality software unless high quality specifications are used. It is thus ncessary to identify the characteristics which high quality specifications should satisfy. Furthermore, specifications are created following some methodology and using some language. Frequently these methodologies and specification languages are ad hoc, thus not necessarily leading to the construction of high quality specifications. It is necessary then to identify the characteristics which methodologies and the specification languages must possess in order to produce high quality specifications.

In this paper a specification quality model is outlined. The objectives of this model are twofold. First, the model should serve as a means to evaluate the quality of a given specification. Second, the model should serve as a means to specify the requirements of methodologies and specification languages which lead almost naturaly to quality specifications.

KEYWORDS AND KEYPHRASES

Metrics, methodologies, quality evaluation, quality prediction , software quality, specification languages, specification quality

RESUMO

A qualidade do software depende diretamente da qualidade de sua especificação. É virtualmente impossível obter-se software de boa qualidade a partir de especificações inadequadas ou mesmo inexistentes. Especificações são criadas seguindo alguma metodologia. Estas metodologias e as linguagens de especificação que usam tendem a ser "ad hoc", levando pois a especificações de baixa qualidade.

Torna-se necessário então determinar as propriedades a serem satisfeitas por especificações, linguagens de especificação e metodologias. Neste artigo examinamos um modelo de especificação e avaliação da qualidade de especificações. Além de servir para avaliar a qualidade de especificações, o modelo serve também para identificar os requisitos a serem satisfeitos por metodologias, linguagens e especificações.

PALAVRAS E FRASES CHAVE

Controle de qualidade, especificação de requisitos de qualidade, linguagens de especificação, metodologias de especificação, qualidade de especificação, qualidade de software

1. INTRODUCTION

Software engineering, though evolving very fast, is still more a craft than a true field of engineering. Several aspects, such as system architecture, human engineering and others, will probably remain as an art. In other words, several aspects of software specification and design will continue relying on individual creativity. In traditional fields of engineering this has remained so, in spite of the better understanding of basic facts and the empirical identification of rules to be obbeyed. Nothing leads us to the conclusion that software engineering will be different.

In traditional fields of engineering though, once an overall design (architecture) has been accomplished, detailed design and manufacturing is usually able to lead to a product performing as described in its specification. Of course, very often prototypes and/or simulation models are developed in order to validate these specifications. However, once specified, there is a high degree of confidence that the product will behave as expected and that it will possess the predefined level of quality. Furthermore, it is reasonable to expect that people manufacturing a given product will understand its specification, in spite of a far lower level of training of these people when compared to the specifiers and designers.

Unfortunately in software engineering we have not reached this level of technical expertise yet. Several factors contribute to this state of the art in software engineering:

1. many software systems are far more complex than most products and systems developed in other fields of engineering;
2. properties such as software reliability usually do not leave margin to tolerances. A simple minor fault or oversight may lead to as great a disaster as gross programming errors. In other fields of engineering, as for instance mechanical engineering, components are manufactured to a given tolerance. As long as this tolerance is assured, the resulting quality will usually be satisfactory.
3. there are no satisfactory specification languages in software engineering. In traditional fields of engineering such languages exist, and are well understood by people using them. Furthermore, the degree of training required to read documents written in these languages is, in general, quite low, and consequently the amount of misunderstandings also tends to be low.
4. there is no satisfactory method to specify, predict and measure the quality of software.

Summing up, software engineering is in need of a family of coherent specification and design languages. These languages must be understandable to those who are going to use these documents. They must also be easy to use and to modify. In addition, a method to specify, predict, evaluate and measure the quality of the resulting software is also needed. Finally, software engineering is in need of a set of instruments leading to a higher productivity, to easier prototyping and simulation, and to a more reliable "manufacturing" process.

2. WORKING HYPOTHESIS

Several specification methodologies and languages have been proposed so far. It seems thus a waste of effort to just propose another methodology. The following are several basic requirements which a software development methodology should possess. These requirements have been determined both empirically [Rocha 82b] and through the study of several existing or proposed methodologies [Alford 76, Alford 77, Alford 81, Ambler 77, Bail 79, Balzer 78, Bell 76, Bell 77, Chandersekaran 81, Davis 77, Delisle 81, Discepolo 81, DeMarco 78, Elschlager 79, Gane 79, Gane 80, Glass 79, Goguen 78, Goguen 79, Green 76, Hammer 79, Kant 81, Levitt 80, Liskov 77, Neighbors 81, Peters 78, Prywes 77, Rocha 82a, Ross 77, Schindler 81, Silverberg 81, Stephens 78, Teichroew 76, Teichroew 77, Yourdon 79, Wasserman 79, Wasserman 81a, Wasserman 81b, Wasserman 82, Willis 81].

1. *methodologies must support the whole life cycle, both as a documentation aid and as a construction aid.* Obviously, specifications must describe the behaviour and the expected quality of the product being specified. They must also describe how this product will be examined in order to evaluate its level of quality. Furthermore, there must be a simple way to transform this specification into a viable and sufficiently efficient design, assuring that the product, if built as designed, will achieve the specified level of quality.

   When observing more closely such a design, it turns out that it consists of a set of specifications. Each of these specifications describes a specific component of the original product.

   Design could then be defined as being the selection of components, the definition of the interactions (form and content) between these components and the individual specification of each of these components.

2. *software development methodologies should be based on several different languages, where these languages form a hierarchy.* In this hierarchy, languages at a lower

level are used to design and/or implement a component defined (specified) using languages at higher levels. The lowest level of the hierarchy comprises programming and data definition languages.

3. *each specification language should be designed to describe the product at a given level of detail, making it manageable to the human mind to understand complex systems*. Notice that the same language may be used to describe a given aspect of the product in varying levels of detail. This is the case, for example, in structured languages such as those used to represent data flows.

4. *there must be a semi-automatic means to migrate from one language of the hierarchy to the next one*. There must also be means to follow sequences of specification and design hierarchies in either direction. Although top down development is considered to be the best way to manage complexity, it is a well known fact that software specification, design and implementation seldon follow a purely top down strategy.

   When examining other fields of engineering, modifications occurring during development and manufacturing are also common place. Similarly to what happens when developing software, many of these modifications are of a technical nature and not just a consequence of changing user needs.

5. *each language should be designed to address the problem being described using this language*. Thus languages should be customized to their field of application, to the level of detail of observation and to the people who are going to read the documents written using these languages. Notice that not necessarily all users will have formal training in the usage and reading of these languages; indeed they may not even have technical training in computing.

6. *at the same level of detail, there should be several different languages, each enabling to understand the same problem from different points of view*. Mechanisms must exist allowing the identification of conflicts and inconsistencies among these different views of the same product. For example, data models and functional models usually work at similar levels of detail and are frequently used to describe the same system. However, often there is no well-defined mechanism to check whether both descriptions are consistent.

   Although describing the same product using different languages leads to redundancy, this redundancy is not necessarily a waste of effort, since it contributes to a better understanding of the problem to be solved. It also

contributes to increase our ability to validate specifications.

7. *each language should embody a method to help in controlling quality.* This control should address form and contents of the specifications and designs. The control of content should address both inherent properties (is it understandable?), and predictive properties (if built as specified, will it do what is needed/ expected?).

8. *languages should not harass people, but should allow a great degree of freedom.* If strict controls are required, the best approach is to design tools which perform these controls instead of introducing them into the languages (for example by means of preprocessors). In this way controls are application specific instead of being general purpose. Such controls have proved to be far more effective than controls imposed automatically through language processors. It constitutes just a managerial problem to ensure that all required controls be applied to a given product. For example, in PASCAL it is almost impossible to develop packages manipulating arrays passed as parameters. The reason for this is the strong typing requirement and the way PASCAL defines arrays. As a consequence, several ways to bypass this difficulty have been invented. Some of these tricks lead to the loss of control of accesses beyond the effective array boundaries. Thus, paradoxically, the establishment of strict controls may lead to a loss of control.

3. SOFTWARE QUALITY

In order to identify the properties of good specifications, we must first understand software quality since specifications are means to achieve quality software.

Software quality is a consequence of the serviceability of this software. The quality is mainly percieved by the users of this software. Software quality is, however, a consequence of its engineering, and thus, a consequence of the quality of the specifications.

Before constructing we must be able to predict whether the quality levels to be achieved are satisfactory. In other words, we must specify the expected quality levels. After construction we must be able to measure the achieved quality and compare these measurements against the specifications.

The following model allows both prediction and control of software quality. This

model is inspired by the model presented by Mc Call [Mc Call 78]. The model is hierarchical and based on the following basic concepts:

1. *quality objectives*: determine the overall expectations quality software should satisfy;

2. *product quality factors*: are the properties which determine quality of a given software from the user's point of view;

3. *engineering quality factors*: are the properties which determine the quality of the engineering of a given software;

4. *criteria*: define specific aspects to be measured or evaluated, determine the method to be used in order to measure or evaluate these aspects and establish the scale to be used when measuring;

5. *metrics*: are quantitative measures obtained according to a specific criteria;

6. *normalization functions*: define a function to be used in order to attribute a value to a given factor in terms of a set of metrics.

Software quality has five objectives:

1. *Utility*: the software produces timely and useful results. Some of the main factors related to this objective are:
   - satisfaction of user needs and expectations;
   - timeliness of the results;
   - availability;
   - reliability;

2. *Ability to use*: the software is easy to use and has been designed to operate in real environments. Some of the main factors related to this objective are:
   - the software is easy to operate, robust and safe;
   - the required data are easy to prepare and results are easy to understand;
   - no unrealistic assumptions regarding its users are made;
   - the software is protected against accidents and misuse;
   - the software may be corrected quickly and restarted after errors or accidents;

3. *Ability to monitor*: the software and its results may be continually evaluated

with regard to resource consumption and with regard to the quality of the results. Some of the main factors related to this objective are:

- possibility to detect errors such as misuse, incorrect data, system errors, accidents, frauds, etc.
- possibility to continually evaluate the level of achievement of each objective;
- possibility to audit.

4. *Ability to evolve*: the software (documentation, programs, etc.) may be maintained, modified, expanded and/or reused. Some of the main factors related to this objective are:

   - maintainability;
   - flexibility, portability;
   - reusability.

5. *Profitability*: the software consumes no more than a justifiable amount of resources in order to achieve its goals. Some of the main factors related to this objective are:

   - compatibility of costs and benefits, both from a financial and a social point of view;
   - efficient use of resources.

In order to develop software of predictable quality levels, specifications should determine acceptable levels of quality this software should possess. They should also determine how this quality will be measured. This may be accomplished by establishing the degrees of each of the quality factors. These degrees vary according to the relative priority or importance of each of the software quality objectives. As already mentioned, factors are evaluated by means of normalization functions and in terms of metrics obtained in accordance with their respective criteria. These quantitative measures yield quantitative estimates for each of the quality factors. The resulting values are then compared with the specified minimum acceptable levels. If a given factor falls below its acceptance level, the product must be improved in order to reach this level.

It is beyond the scope of this paper to detail further this model. We refer the interested reader to [McCall 78, Gilb 77, Boehm 78, Peercy 81, PSS 81, ACM 78].

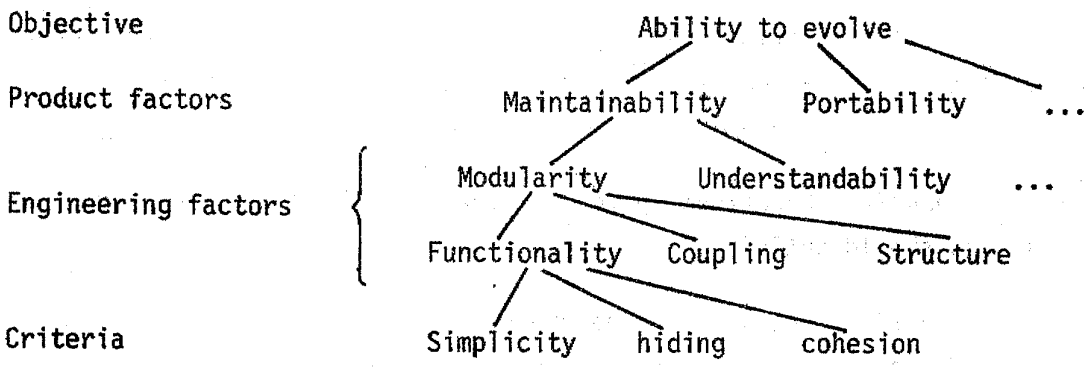

Figure 1. Sample portion of the software quality model

Criterion simplicity

. Definition: evaluates the degree of unnecessary complexity of the module's implementation

. Evaluation method: estimation according to scale

. Scale 10: module implements exactly what has been specified using the simplest known satisfactory algorithm

9: module implements exactly what has been specified using an unnecessarily complex algorithm

6: module implements more functionality than necessary to implement the specified function

3: module requires more interface than necessary to implement the specified function

Figure 2. Sample of a criterion definition

Figure 1 and 2 illustrate the quality specification and control model.

4. SPECIFICATION QUALITY

As mentioned before, the development of quality software depends on the quality of its specifications. It should be clear that a systematic approach should be taken in order to identify the properties good specifications should possess. Unfortunately there is no sufficient theoretical basis against which a set of properties might be

tested. In the sequel, we outline a model for specification quality. Structurally this model is similar to the one used to evaluate the quality of software. Following are the additional concepts used:

1. *Specification quality objectives*: determine the overall expectations which specifications should satisfy;

2. *specification quality factors*: are the properties which determine the quality of a given set of specifications, from the point of view of the user of these specifications (e.g. quality control, software verification and validation, design, etc.);

3. *specification construction quality factors*: are the properties of the construction process (methodology) and the tools used during this process (e.g. languages).

Notice that specification quality is a means to achieve engineering quality. Engineering quality depends also on the process and on the instruments by means of which these specifications and designs are built. It finally depends on the management process employed during construction and maintenance. This latter aspect, though very important, will not be examined further in this paper.

Specifications should satisfy the following objectives:

1. *Ability to evolve*: it must be easy to construct specifications. It must also be easy to add details to given specifications when designing a solution, even if this addition is achieved using a different language than the specification language originally used;

2. *Verifiability*: it must be easy to evaluate the inherent properties of specifications (specification construction);

3. *Validability*: it must be easy to evaluate the performance of the product being specified. It must be possible to do evaluation in a predictive manner, that is, it must be possible to estimate the performance of the product within known tolerances before its construction. It must also be possible to examine whether the product performs as expected once it has been constructed, that is, it must be possible to evaluate the effective quality of the product;

4. *Modifiability*: it must be easy to modify already existing specifications, designs and programs.

In what follows we will outline a list of specification quality factors and criteria. This list is not intended to be exaustive. We will also not discuss the measuring processes and scales related to each of the criteria identified.

1. *adequateness*: the specification truly represents what is understood as being the user's needs and expectations.

   a. *conceptual correctness*: whether all specified aspects satisfy the user's needs and expectations.

   b. *conceptual necessity*: whether all specified aspects must effectively be present.

   c. *conceptual completeness*: whether all aspects which should be specified have effectively been specified.

   d. *consistency*: whether there are no contradictions among specified aspects, both within the present specification and within the collection of specifications produced so far.

   e. *conceptual agreement*: whether all specified aspects exactly represent what is perceived by the group of specifiers.

2. *Reliability*: the specification is intrinsically correct.

   a. *formality*: the rigour and precision of each aspect's specification.

   b. *formal correctness*: whether all specified aspects have been specified in a formally correct fashion, e.g. the specification methodologies and language have been correctly used.

   c. *non-ambiguity*: whether there is just one possible interpretation for each specified aspect.

   d. *consistency*: whether there are no formal contradictions among specified aspects whithin the present specification, or within the set of specifications and designs produced so far.

   e. *explicitness*: whether all aspects have been explicitly specified; no aspects should be defined by context.

   f. *adherence to standards*: whether the specification adheres to the established standards.

3. *feasibility*: it should be possible to implement the product being specified from the technical, economical and administrative points of view.

   a. *economical feasibility*: whether the specified product may be built whithin the existing economic restrictions (e.g. costs, benefits, opportunity, social impacts, etc.).

   b. *technical feasibility*: whether the product may be constructed within the existing technical restrictions (e.g. response times, memory consumption,etc.).

   c. *administrative feasibility*: whether the product may be built within the existing administrative restrictions (e.g. manpower, computer time, calendar time, etc.).

4. *Adherence to stepwise refinements*: it should be possible to produce a design for the current specification and the collection of designs and specifications produced so far should obey stepwise refinement rules.

   a. *non restrictiveness*: whether the specification imposes no unnecessary restriction on the selection of design alternatives.

   b. *uniformity of detail*: whether all specified aspects are treated at the same level of detail.

5. *Understandability*: it should be easy to understand what has been specified. Notice that this factor depends on the reader. This imposes a set of restrictions on the languages used, if the specification is to be read by people who have very little training in computing.

   a. *communication modularity*: whether the specification defines a single product considering the current level of detail (cohesion), and whether the interfaces of the present specification with other specifications are well defined (coupling).

   b. *uniformity*: whether the terms and symbols are uniformly used within the set of specifications.

   c. *conciseness*: whether the amount of information contained within the present specification reasonably corresponds to its physical size.

   d. *non-redundancy*: whether no aspect is specified more than once at the same level of detail and from a similar point of view. Redundancy should not occur within the present specification, nor should it occur within the set of

specifications produced so far.

. *Manipulatability*: it should be easy to access and manipulate the set of specifications produced so far.

a. *availability*: whether a specification is present whenever required.

b. *accessability*: whether it is possible to localize and access a given specification whenever required.

c. *structure*: whether the set of specifications produced so far is well organized, allowing the traversal and access to the several documents already produced. It should be possible to traverse the hierarchy of documents in either direction.

d. *traceability*: whether it is possible to follow the evolution of a given aspect or requirement among the different specifications produced so far.

adequateness	reliability
conceptual correctness	formality
conceptual necessity	formal correctness
conceptual completeness	non-ambiguity
consistency	consistency
conceptual agreement	explicitness
	adherence to standards
feasibility	adherence to stepwise refinements
economical feasibility	non restrictiveness
technical feasibility	uniformity of detail
administrative feasibility	
understandability	manipulatability
communication modularity	availability
uniformity	accessability
conciseness	structure
non-redundancy	traceability

Table 1. Specification quality evaluation factors and criteria

Table 1 summarizes the factors and criteria to be used when evaluating the quality of specifications.

The following factors and criteria are related to the quality of the construction process (methodology).

7. *Applicability*: the methodology should be applicable to a large set of different problems and it should support the whole life cycle of the software.

   a. *life span*: whether the methodology supports the whole life cycle of the software being constructed (and maintained).

   b. *generality*: whether the methodology applies to a large set of different applications.

   c. *adaptability*: whether the methodology may easily be adapted in order to apply to a new application.

8. *Organization*: the methodology should contribute to organize thinking and expression.

   a. *hierarchy*: whether the methodology allows hierarchical thinking and expression.

   b. *modularity*: whether the methodology allows modular thinking and expression.

9. *Support*: the methodoloty should provide ample and effective support to the tasks which are realized during the construction of a given specification.

   a. *specification construction support*: the quality and effectiveness of the methodology with regard to its support of specification construction.

   b. *validation support*

   c. *verification support*

   d. *modification support*

   e. *management support*

In table 2 we summarize the factors and criteria to be used when evaluating the quality of a specific methodology.

Aplicability	Organization
life-span	hierarchy
generality	modularity
adaptability	
Support	
specification construction support	
validation support	
verification support	
modification support	
management support	

Table 2. Specification methodology quality evaluation factors and criteria

The following factors and criteria determine the quality of specification languages:

10. *Useability*: the specification language should be easy to learn and use.

    a. *documentation*: the existence, clarity and accessibility of the documentation describing the specification language.

    b. *educational requirement*: the level of formal education required both for specifiers and for readers of specifications using the language.

    c. *training requirement*: the amount of training and practical experience required for specifiers and readers in order to make effective use of the specification language.

    d. *naturalness*: whether the specification language is conceptually close to the problem being solved.

    e. *evolution aids*: the existence and effectiveness of the available aids to further detail already specified aspects.

    f. *ease to automate*: the possibility to develop and use automated tools supporting the specification language.

    g. *indexing support*: the existence and effectiveness of aids to construct cross-references and tables of contents.

11. *evaluation aids*: the language should aid the verification and validation of specifications created using this language.

a. *formality*: the degree of formality of the specification language.

b. *verification support*: the existence and effectiveness of the instruments supporting the verification of specifications written in this language.

c. *validation support*: the existence and effectiveness of the instruments supporting the validation of specifications written in this language (e.g. prototyping support).

Useability	evaluation aids
documentation	formality
educational requirement	verification support
training requirement	validation support
naturalness	
evolution aids	
ease to automate	
indexing aids	

Table 3. Specification language quality evaluation factors and criteria

In table 3 we summarize the factors and criteria to be used when evaluating the quality of a specific specification language.

5. CONCLUSION

The model outlined above may seem quite complex when examined at first sight. However, this is not necessarily true. The construction of a specification quality control model is a one-time effort, applying to all specifications produced later on. The quality control of methodologies and of specification languages is performed only when selecting and/or defining such methodologies and languages. The amount of set-up work is thus comparatively low when considering the life span of use of methodologies and languages.

In practice the model will tipically be a check list. With some training, quality inspectors will be able to obtain metrics according to the different specification quality criteria with a reasonably small amount of effort and time. Furthermore, as a side effect of the existence of such an evaluation model, analysts and designers

will tend to produce high quality specifications, since now they know what the acceptance criteria are.

The present model has been used in an industrial environment and proved to be effective at low cost [Barros 82]. The evaluation method used was very close to the method described in [Peercy 81]. Obviously several more experiments must be conducted in order to assess and tune criteria and normalization functions.

Much work must still be done in order to develop a workable model. First of all, measuring processes and scales must be developed for each of the criteria. Furthermore, the model must be examined with regard to completeness. The latter seems to be quite difficult since there is very little theoretical basis. Consequently, we are using a learning process, collecting and cataloguing as many factors and criteria as possible. This collection is then critically examined in order to eliminate redundancies and to identify properties which have not adequately been addressed yet.

Although being empirical, it is our firm belief that the model is technically and economically feasible, and that the resulting software will be of a far higher degree of quality than what is usually achieved in current practice. Furthermore, it is our belief that new specification methodologies and languages should be designed only after a better understanding of their quality requirements has been acquired.

REFERENCES

[ACM 78] ACM Sigmetrics

Proceedings of Software Quality and Assurance Workshop; ACM SIGMETRICS, vol. 7, nºs 3 and 4; November 1978.

[Alford 76] Alford, M.W.; Burns, I.F.

"R-Nets: a graph model for real-time software requirements"; in *Proceedings of the Symposium on Computer Software Engineering*, New York; 1976.

[Alford 77] Alford, M.W.

"A requirements engineering methodology for real-time processing requirements"; in *IEEE Transactions on Software Engineering*, vol. SE-3, nº 1; Jan. 1977.

[Alford 81] Alford, M.W.

"Experience with the software development system"; in *Software Engineering Environments*; Hunke, H. ed.; North-Holland; 1981.

[Ambler 77] Ambler, A.L. et al
"GYPSY: A Language for specification and implementation of verifiable programs"; in *Proceedings of the ACM Conference on Language Design for Reliable Software*; March 1977.

[Bail 79] Bail, W.G.
"User experience with specifications tools", in *ACM SIGSOFT Software Engineering Notes*, vol. SE-2, nº 3; July 1979.

[Balzer 78] Balzer, R.; Goldman, N.; Wile, D.
"Informality in program specifications"; in *IEEE Transactions on Software Engineering*, vol. SE-4, nº 2; March 1978.

[Barros 82] Barros, J.C.C.
A Specification Quality Evaluation Model; M.Sc. Dissertation, Computer Science Department, Universidade Federal de Pernambuco, Recife, Brazil, 1982 (in Portuguese).

[Bell 76] Bell, T.E.; Bixler, D.C.
"A flow-oriented requirements statement language"; in *Proceedings of the Symposium on Computer Software Engineering*; New York; 1976.

[Bell 77] Bell, T.E.; Bixler, D.C.; Charles, R.
"The software development system"; in *IEEE Transactions on Software Engineering*, vol. SE-3, nº 1, January 1977.

[Boehm 78] Boehm, B.W. et alii
Characteristics of Software Quality; North Holland; 1978.

[Chandersekaran 81] Chandersekaran, C.S.; Linder, R.C.
"Software specifications using SPECIAL language", in *The Journal of Systems and Software*, vol. 2, nº 1; February 1981.

[Davis 77] Davis, G.; Vick, C.
"The software development system"; in *IEEE Transactions on Software Engineering*, vol. SE-3, nº 1; January 1977.

[Delisle 81] Delisle, N.M. et al
"Tools for supporting structured analysis"; in *Automated Tools for Informations Systems Design*; Schneider, H.J.; Wasserman, A.I. eds.; North-Holland; Amsterdam; 1982.

[Discepolo 81] Discepolo, A.G.
"Towards a practical specification language"; in *ACM'81 Conference Proceedings*;

November 1981.

[DeMarco 78] De Marco, T.
"Structured Analysis and System Specification"; Yourdon; New York; 1978.

[Elschlager 79] Elschlager, R.; Phillips, J.
Automatic Programming; Computer Science Department, Stanford University, Report STAN-CS-79-758, August 1979.

[Gane 79] Gane, C.; Sarson, T.
Structured Systems Analysis: Tools and Techniques; Prentice-Hall; 1979.

[Gane 80] Gane, C.
"Data design in structured systems analysis"; in *Tutorial on Software Design Techniques*; Freeman, P.; Wasserman, A.I. eds. IEEE Computer Society; 1980.

[Gilb 77] Gilb, T.
Software Metrics; Whinthrop; 1977.

[Glass 79] Glass, R.L.
Software Reliability Guidebook; Prentice-Hall; Englewood Cliffs, New Jersey; 1979.

[Goguen 78] Goguen, J.A.; Thatcher, J.W.; Wagner, E.G.
"An initial algebra approach to the specification, correctness and implementation of abstract data types"; in *Current Trends in Programming Methodology*, vol. IV; Yeh, R. ed.; Prentice Hall; 1978.

[Goguen 79] Goguen, J.A.; Tardo, J.J.
"An introduction to OBJ: a language for writing and testing formal algebraic program specifications"; in *Proceedings Specifications for Reliable Software*; 1979.

[Green 76] Green, C.
"The design of the PSI program synthesis system"; in *2nd International Conference on Software Engineering*; 1976.

[Hammer 79] Hammer, M.; Ruth, G.
"Automating the software development process"; in *Research Directions in Software Technology*; Wegner, P. ed.; MIT Press; 1979.

[Kant 81] Kant, E.; Barstow, D.
"The refinement paradigm: the interaction of coding and efficiency knowledge in program synthesis"; in *IEEE Transactions on Software Engineering*, vol. SE-7, nº 5; September 1981.

[Levitt 80] Levitt, K.; Robinson, L.; Silverberg, B.
"Writing simulatable specifications in SPECIAL"; in *The Use of Formal Specification of Software*; Berg, H.K.; Giloi, W.K. eds.; Springer; 1980.

[Liskov 77] Liskov, B.; Zilles, S.
"An introduction to formal specifications of data abstractions"; in *Current Trends in Programming Methodology*, vol. I; Yeh, R. ed.; Prentice-Hall; 1977.

[McCall 78] McCall, J.
"An introduction to software quality metrics"; in *Software Quality Management*; Cooper, J.D.; Fisher, M.J. eds.; Petrocelli; 1978.

[Neighbors 81] Neighbors, J.M.
Software Construction using Components; Ph.D. Thesis; Department of Information and Computer Science; University of California, Irvine; 1981.

[Peercy 81] Peercy, D.E.
"Software maintainability evaluation methodology"; in *IEEE Transactions on Software Engineering*, vol. SE-7, nº 4; July 1981.

[Peters 78] Peters, L.
"Relating software requirements and design"; in *Proceedings of the Software Quality and Assurance Workshop*; November 1978.

[Prywes 77] Prywes, N.S.
"Automatic generation of computer programs"; in *Advances in Computers*, vol. 16, Rubinoff, M.; Yovits, M. eds.; Academic Press; 1977.

[PSS 81] Perlis, A.J.; Sayward, F.G.; Shaw, M. editors
Software Metrics; MIT Press, Cambridge, Mass.; 1981.

[Rocha 82a] Rocha, A.R.C.; Staa, A.V.
"Software Quality in the Requirements and Specification Phase"; in *The Fourth Israel Conference on Quality Assurance*; Herzlia, Israel; October 1982.

[Rocha 82b] Rocha, A.R.C.; Staa, A.V.
"Towards a family of specification languages"; in *Revista Brasileira de Computação*, vol. 2; 1982 (in Portuguese).

[Ross 77] Ross, D.T.
"Structured analysis (SA): a language for communicating ideas"; in *IEEE Transactions on Software Engineering*, vol. SE-3, nº 1; January 1977.

[Schindler 81] Schindler, M.
"Today's software tools point to tomorrow's tool systems"; in *Eletronic Design*; July 1981.

[Silverberg 81] Silverberg, B.A.
"An overview of the SRI hierarchical development methodology"; in *Software Engineering Environments*; Hunke, H. ed.; North-Holland; 1981.

[Stephens 78] Stephens, S.A.; Tripp, L.L.
"Requirements expression and verification aid"; in *Proceedings of the 3rd International Conference on Software Engineering*; May 1978.

[Teichroew 76] Teichroew, D.; Hersley, E.A.
Computer Aided Structured Documentation and Analysis of Information Processing Systems Requirements; ISDOS Project, University of Michigan; August 1976.

[Teichroew 77] Teichroew, D.; Hersley, E.A.
"PSL/PSA: A computer aided technique for structured documentation and analysis of information processing systems"; in *IEEE Transactions on Software Engineering*, vol. SE-3, nº 1; January 1977.

[Yourdon 79] Yourdon, E.; Constantine, L.L.
Structured Design; Prentice Hall; 1979.

[Wasserman 79] Wasserman, A.I.; Stinson, S.K.
"A specification method for interactive information systems"; in *Proceedings of Specifications for Reliable Software*; 1979.

[Wasserman 81a] Wasserman, A.I.
"User software engineering and the design of interactive systems", in *5th International Conference on Software Engineering*; March 1981.

[Wasserman 81b] Wasserman, A.I.
The User Software Engineering Methodology: an Overview; University of California, San Francisco; Technical Report 56; 1981.

[Wasserman 82] Wasserman, A.I.
"Automated tools in the information system development environment"; in *Automated Tools for Information Systems Design*; Schneider, H.J.; Wasserman, A.I. eds.; North-Holland; 1982.

[Willis 81] Willis, R.R.
"AIDES: computer aided design of software systems-II"; in *Software Engineering Environments*; Hunke, H. ed.; North-Holland; 1981.