Evaluation and selection criteria for software requirements specification standards

E.A. Giakoumakis and G. Xylomenos

***Abstract*— Various organisations have published proposals to prescribe the form and content of software requirements specification documents; the standards were designed to support the specific needs of these organisations and the intricacies of their development projects. To help third parties in taking advantage of this body of work, a set of criteria are proposed and discussed that can be used to evaluate such standards, according to the unique characteristics of specific combinations of organisations and software development projects, and then the question of how the criteria can be applied in an evaluation, selection and tailoring process, depending on the circumstances, is discussed. Finally, the criteria are demonstrated by applying them on some published standards, to help interested organisations to preselect those that seem most appropriate for their needs.**

I. INTRODUCTION

Irrespective of the development process that an organisation follows while building a computerised system, the specification of the requirements for the software component of the system always stands out as a distinct and important task. By applying common industrial practice, software development is a process that consists of a number of phases that construct and refine the final product. One definition for the software requirements specification phase in particular is that it is a procedure of detailed specification of the functions that the software is to perform in the system, as well as the constraints under which it will operate [1]. Even though the importance of this phase has been proven in many cases where insufficient efforts dramatically increased development costs [1] or has even led to the abandonment of projects [2], requirements engineering still seems to be considered by practitioners as more or less an art form.

This paper divides the software requirements analysis phase into the activities of gathering and recording requirements, more commonly called analysis and specification, concentrating on the development and application of engineering principles and methods to the latter task. It is motivated by the proliferation of standards for writing a software requirements specification document (SRSD) published by various organisations. Our goal is to help other organisations evaluate, select and customise one of these standards and then use it in support of their individual projects and needs. In the following we will see that there is no overall winner in the standards arena, as the needs of organisations working on different projects can, and do, vary. Thus, selection and application of a documentation standard remains a challenging but, as we will also show, important task.

E.A. Giakoumakis is with the Department of Informatics at the Athens University of Economics and Business, Patission 76, Athens 10434, Greece; G. Xylomenos is with the Department of Computer Science and Engineering at the University of California, San Diego, La Jolla, California, 92092-0114, USA.

II. SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENTS

A. Nature and uses of specifications

The requirements stated in the SRSD pertain only to the software part of a system; in a preceding phase, requirements for the system as a whole have been specified, with some of these requirements having been allocated to the software. These system-wide specifications are the first input for the software requirements specification phase; the second input is information gathered from the future users of the system under development. Using the latter, the requirements stated at the system level are elaborated to become a description of software behaviour and constraints on its operation. This elaboration process is the requirements analysis part of the phase; these requirements are recorded on some appropriate form, in the requirements specification part of the phase.

To see why each software requirement is so important, we should further clarify what it is. An appropriate definition from [3] is that it is ‘a software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document’. The essence of the definition is that a requirement is a contract among the various parties involved in development: if the software behaves according to the requirements specified in the SRSD, then it should be acceptable to its users. Naturally, the users will not accept that the software satisfies their needs simply because it does what the SRSD says, but the liability of the developer stops there: it is the responsibility of the user organisation to ensure that the SRSD actually specifies software that will be acceptable to its users, a difficult and time consuming task. At the same time, early agreement between specified and actual requirements results in fast development, while late reconciliation of discrepancies among them not only slows down development, but also immensely raises its costs.

The importance of the SRSD arises from the fact that it is the first document to describe the software under development in detail. As such, it is used in practically every subsequent activity, being at the same time a statement of the user’s needs, a statement of the requirements for the implementation and a reference point during maintenance [4]. A similar view is that it is an agreement on the requirements, a basis for design and a reference point during validation [5]. Thus, the SRSD is both an agreement among users and developers, as it is used to check whether the software behaves as requested, and a common point of reference between analysts, designers, implementers, testers and maintainers working on the project. Even very small in-house development projects will have to refer to the SRSD long after it has been written, due to future needs for maintenance, optimisation and capability extensions.

The importance of software requirements is stressed even in basic software engineering textbooks [1], [6]; there is also considerable work especially on requirements gathering and specification, both practically [7] and theoretically [8] oriented. Still, requirements analysis remains an activity that needs skill and experience, hence the 'art form' characterisation in Section I. As requirements are eventually recorded in the SRSD, it is more accurate to think of the problems of composing an SRSD as dealing with both its content and its form; that is, the information contained in the document and the organisation and presentation style of this information.

B. Form, content and standardisation of specifications

Consider what happens when the requirements engineers have formed an idea of the software requirements. The optimal way to record this information, will generally depend on its content. If the project is large, this information may have to be organised hierarchically and split into modules that communicate using strict interfaces, to ease the understanding, use and maintainance of the SRSD. This activity may also be required if the development proceeds in multiple design phases or if many subcontractors are developing parts of the software. In contrast, the overheads imposed by this distributed organisation would be redundant for a small project, messing up rather than improving the requirements. Depending on the backgrounds and roles of its users, the structure and presentation of the SRSD may need to accommodate different conventions, jargons and standards; thus, organisations involved in development are another factor to be considered.

As a concrete example of the difficulties involved in composing an SRSD under different circumstances, consider first a comprehensive accounting system developed outside the user organisation: the SRSD should describe accurately the functional aspect of the software, while paying less attention to non-functional requirements such as performance goals. The format and content of the SRSD should suit the needs of both users and developers, with appropriate parts of the document following the established conventions of each party. Contrast this with a small real-time software component developed in-house: there non-functional requirements such as timing constraints and response pairs would take up most of the SRSD to ensure that the software will correctly interact with other components of the system, while the language and conventions used in the document would certainly be understandable to all people involved. These observations imply that the project, the development process, the organisations involved, and generally the required content of the SRSD as determined by these factors, should influence the form of the SRSD to maximise its effectiveness.

A standard is an 'approved, documented and available set of criteria used to determine the adequacy of an action or an object' [9]; this definition does not help us much, though, since an optimal way to perform requirements analysis and specification has not yet emerged. A standard for an SRSD lies somewhere between the definition above and a guide for specifying requirements. Its usual form is a content outline, annotated so that its authors will be able to find where to record each kind of requirement and, conversely, its users will be able to find where these requirements are. Rigidity in the outline varies considerably among standards, providing more or less flexibility. On the other hand, the annotations for the contents are usually just guidelines of what should be included in each section, and sometimes how to present it. Note that some standards are parts of complete frameworks for all development phases, covering anything from documentation to development processes and procedures.

By applying a documentation standard while composing an SRSD, its authors can increase their productivity by following a familiar outline, using it as a checklist during requirements analysis to see what should be considered for inclusion, while users of the SRSD will be able to find directly what they need once they have become familiar with the outline. It can also serve as a checklist for those responsible for reviewing the SRSD for completeness; even managers allocating tasks to development teams and monitoring their progress can base such activities on this checklist. Thus, a documentation standard is both a guide and an interface among the various parties involved in the creation, use and maintenance of the SRSD. Not all interfaces are of the same value though, as it is not enough to define any strict interface, as is done with other artefacts: the guidebook aspect of a standard for a process as challenging as requirements analysis highlights the importance of the interface's quality.

Since adoption of a standard can have such beneficial results, the question is what it should look like. We saw that the content of a document influences its optimal form, and this is particularly true for different projects involving different organisations, but customising an SRSD is not easy when a standard must be adhered to. In fact, the adoption of a specific documentation standard establishes an influence from the form of the SRSD to its content: the standard at least imposes a structure on requirements and it may also prioritise their importance, by including more or less detail about some aspects of the software. This is a direct consequence of using the standard as both a content and a structure guide. Once a standard has been chosen for one or many projects, this interface, for the most part, must freeze. If we deviate significantly from the standard with the intention of optimising a specific SRSD, we lose the benefits of the standardised interface. Standardisation thus comes at a cost, not related to its application, an activity requiring only staff training, but caused by enforcement of the standard on diverse projects.

This reverse influence from form to content seems undesirable, since it should be the content that matters and imposes an optimal structure on the presentation. However, for a document used by so many people and at so many different times, the benefits from standardisation are too important to be neglected: communication among people at different times is eased tremendously, and the burden of structuring the SRSD is removed from the requirements engineers. The proliferation of such standards, coming from quite diverse organisations, shows clearly that these benefits are understood, but an examination of them (as we make in Section IV reveals that the diversity in their sources is distinctly reflected on the standards themselves.

Our purpose in this paper is not only to point out the pitfalls of standardisation and provide general guidelines for avoiding them; we intend to demonstrate how an organisation can evaluate an available documentation standard according to its needs, using a set of well defined criteria in a disciplined way, and then further tailor it to best serve its goals.

III. Evaluation and Selection Criteria

A. Design Principles

An organisation may use a standard either because it is required by other involved parties or because it believes that it is beneficial for development. In some cases, the standard to be used will be enforced; this applies to contractors of the US Department of Defence. In most cases though, the organisation will be free to choose the best standard for its specific needs. One option is to develop a standard from scratch, taking into account the given situation during design. Although this may provide the best results in terms of quality and suitability for both organisation and project, such a process requires extensive experience, time and funds. Furthermore, deployment of the standard will require training of all present and future personnel involved with development, even if some people have had already experience of other standards.

As a case study in independent standards development consider the US Department of Defence, one of the biggest software consumers in the world. Its standards for automated information systems have evolved from the 7935 standard in 1983, through the 7935A [10] in 1988, culminating into the 498 [11], [12], [13] standard in 1994. For its embedded systems on the other hand, the original 2167 standard was published in 1985, replaced by 2167A [14], [15], [16] in 1988, and was finally merged with 7935A into the 498 proposal. This decision shows the importance of using a single standard for all projects in an organisation, despite their diversity. The comments accompanying standard 498 [17] illustrate how difficult it is to design and freeze a standard as needs change, projects evolve and technology advances, during a period of more than 10 years.

This example shows that for most organisations it will be uneconomical, or even impossible, to invest the time and money required to design a standard from scratch. Considering the plethora of available standards, we focus in this paper on how an organisation should choose an existing standard for use. To make an informed choice, an organisation must first understand the intricacies of the proposed standards and evaluate them with an eye on its needs and on the nature of the projects that it is involved with. Then, it may select the most appropriate standard for either a single or a set of projects, and finally, it can proceed to tailor this standard, on a global or a case-by-case basis. We emphasise tailoring following the 498 proposal [11] which accepts that no standard can be both rigid and appropriate for all circumstances, either because standards are vague, or because modifications make the standard more suitable for the projects at hand. Tailoring will be facilitated by our criteria, since any problems revealed while evaluating a standard will be the first targets for modifications. This also implies that an evaluation process can even help in cases where a standard has been externally enforced.

Our design principle in formulating a set of evaluation criteria is that these criteria should be able to uncover all the relevant advantages and disadvantages of a standard with respect to its ability to serve as the basis of a good SRSD. An SRSD is good if the quality of the requirements that it contains is high and the way they are presented in the SRSD satisfies the needs of its users. If we look at the qualities that a good SRSD should possess [18], we will see that some of them relate directly to the quality of the requirements and are relatively orthogonal to their presentation; on the other hand, achieving some other qualities can be facilitated by the use of an appropriate standard. As we want the criteria to be helpful both for selection and tailoring in any situation, we must try to cover as many aspects and views of the SRSD as possible, understanding that not all of these details will be relevant for all circumstances. We will defer the discussion on application of the criteria on specific organisation-project combinations to Section III-J; each criterion includes in its description comments on its applicability under different assumptions, to ease selection of the appropriate set of criteria for the circumstances at hand.

Although the SRSD's users are our focal point, we must not forget that their needs vary according to many factors: the organisations involved, the people employed and the specific details of projects under development, form a complex environment for the evaluation criteria. Furthermore, the different needs that the users of the SRSD have, depending on their roles in its development and use, further increase the viewpoints from which we should look at the standards. Some aspects of a standard, such as its completeness in coverage of all types of requirements, inclusion of all software features [6], and explanation of all concepts mentioned in the SRSD [4], [18], are interesting in all cases, because they directly influence the quality of the requirements. Other aspects can be more interesting in some circumstances than in others: examples include adequate coverage of the system of which the software is a component and the operating environment of the software, easy handling of modifications and additions, and low coupling of the SRSD's contents [19]; a form of presentation that conveys information in layers of detail [6] to help its users locate fast what they need; support for traceability of requirements in the SRSD [18]; definition of external software behaviour only, ease of use of the SRSD for reference and having the same meaning to all its users [1], [6].

In the following we define such a set of criteria, based on the aspects examined above, the study of various existing standards and experience with application of these standards on real projects. For each criterion we present a discussion on its nature, its applicability and usefulness, show our motivations in defining it, and then give a sample scale of evaluation for the standards according to it. In Section IV we apply the criteria, using the sample scales, to analyse and evaluate a set of existing standards. Both presentation of the criteria and standards evaluation try to capture as many views of the standards as possible. We deal in Section III-J with the matter of using the criteria to support selection and tailoring for specific circumstances.

B. Independence

This criterion examines two related aspects of a standard: whether it has been designed so that it can be used only in conjunction with specific requirements analysis methodologies and/or software life cycle models; the former is a local constraint for the analysis phase while the latter is a global constraint on the project as a whole. It may also be the case that a standard not imposing the local constraint, indirectly enforces the global one, because its use implies the adoption of other standards that in turn enforce some development practices. We should not confuse integration (see Section III-C) with independence: a standard may belong to a complete series without it having to be used in any specific way; restrictions lie in the methods enforced, if any, and not in using other standards, since a flexible series may be able to accommodate different methodologies in any case.

Such constraints are inherent in older standards [10], especially ones that were developed with a specific user community in mind, which had somewhat stabilised around a set of practices. Some standards mention that they are not suited for specific models [18], but the trend seems to be towards accommodation of multiple methodologies and models using the same standard [11], [17]. A standard depending on specific methods of development could be frowned on, as there is still no method widely accepted as the best for all circumstances. Thus, such restrictions may lead to significant problems when the chosen method is ill-suited to the projects at hand. As life cycle models emerge and requirements analysis methodologies evolve, even a very good attempt to freeze a standard around a successful combination of the two will soon be outdated and will have to be replaced. The motivation for designing such a restricted standard is the hope that by early adoption of a specific set of methods, precision (see Section III-D will be enhanced and documentation needs will be better supported; in contrast, a standard designed to be applicable to many methods will unavoidably be more vague in many respects and will need to be carefully customised in each case.

It should be clear then that accommodating widely different methodologies comes at a price: the design and use of a methodology independent standard should exercise extreme care to avoid leaving the standard so vague that no two of its users will agree on its correct use. Tailoring a standard to fit a specific methodology is not only a challenging process, but it may also lead to losing the benefits of standardisation due to arbitrary modifications. A well-designed standard that was built around a methodology closely matching an organisation-project pair could, in principle, be a good idea when an organisation is involved with fixed projects. In practice, such restrictions do not seem to be justified, and only obsolete standards [10] seem to advocate them.

An evaluation scale for independence could consist of three ratings. High independence would be associated with standards completely decoupled from specific methodologies, while low independence would be associated with standards indirectly tied to specific methodologies or directly built over a fixed set of methods; such standards should be considered inappropriate for use outside their intended field. The middle ground is covered by standards demonstrating medium independence: these may have been influenced by a methodology or they may not be able to co-exist with some methodologies at all, but they could be modified in a straightforward manner to extend their scope, if their other merits compensate for the risks associated with tailoring.

C. Integration

The integration criterion examines whether the standard at hand is part of a complete series of documentation standards, covering all development phases. An organisation may favour the adoption of a complete series, expecting enhanced uniformity among documents and hoping that it may be used without additional effort to make the different standards for each stage co-exist with each other. Uniformity can lead to easier movement among pieces of the project documentation, improving traceability of requirements (see also Section III-I), but only if similar design principles were used when composing the standards themselves. Besides expending significant resources to reconcile differences among standards from various sources to achieve the uniformity of an integrated series, an organisation runs the risk of having the standards deviate so much from their original definitions, that they are not standard any more. Of course, even an integrated series is not guaranteed to be immediately applicable to real projects, as the standards may be too general (see Section III-D), to enhance independence (see Section III-B).

Although decoupling the design of a specific standard from that of a series may result to a optimisations, we do not have any meaningful data to support such a thesis. Some organisations may not be so sensitive to integration, however, either because they are in any case performing a complete tailoring job to accommodate their needs, with reconciliation changes being essentially free, or because they just merely to replace their SRSD standard alone; this may occur when an organisation switches to a methodology not supported by its existing standard. Another case where integration is irrelevant arises when the other standards of the series are inappropriate for the circumstances and a decision against adopting the complete series has been made.

A sample characterisation scale for integration could split standards in those depicting high integration, by being part of a complete series standards which is usable with minor modifications; those that present low integration, by not being part of any series of standards at all; and those that have medium integration, with this latter case applying to standards that are either part of an incomplete series, not covering some development phases, or belong to a series of very vague standards that need substantial customisation before use. Note that if tailoring one standard is considered a difficult and risky task, modification of a complete set of standards multiplies the costs and risks accordingly (see Section III-D).

D. Precision

The precision criterion examines how concisely and thoroughly a standard describes the contents of the outline that it proposes. Precision increases as descriptions become more exact and ambiguities are resolved, so that there can be no dispute of where on the SRSD each kind of requirements should be and, conversely, which requirement belongs to each section. Lack of precision may lead to different interpretations of the SRSD by its authors and users, hindering communication and causing costly misunderstandings. An imprecise standard negates the benefits of standardisation: it must be made more precise, with different versions of it not being compatible, or else its users will make their own assumptions while using it, causing chaos.

A standard usually lacks in precision because its creators did not want to overconstrain its applicability. This can be a valid worry for standards designers, since by being more specific in the description of a standard we may lose in flexibility; past standards [10] indicate that enhanced precision and limited scope of application can coexist. On the other hand, an imprecise standard would be fine [20] if the user organisation were to accept modifications to it that would hardly leave it a standard any more. Existing standards demonstrate that one can be precise enough to avoid ambiguities without overconstraining the standard [11], provided that flexibility and precision are considered early on during design. Fortunately, it seems that the adherence to precision even in early standards [18] has caught on and this attribute is sufficiently valued to be a part of most modern ones. A general method to make a standard be interpreted unambiguously is to provide examples to clarify the description; this does not constrain the standard as much as a rigid and all-inclusive specification, but it still serves as an adequate guideline for the users. Note that an organisation looking for a base to develop its own standards would not at first care much about precision, because heavy tailoring would be needed in any case (see also Section III-B). On the other hand, using a standard lacking in precision will certainly cause lots of problems; it seems strange, to say the least, that there are standards that intend to be usable as they stand but have major shortcomings in this area.

An evaluation scale for the precision criterion could consist of three degrees. A standard has high precision when it is completely clear from its description where each requirement goes; as mentioned above, this characterisation may be obtained either by overconstraining the standard with rigid descriptions or by providing adequate examples for a more flexible description. Low precision is associated with standards that are so vague in their descriptions that are practically impossible to use as they stand. The, both costly and risky, solution of customising such a standard to make it unambiguous would be acceptable only if a decision to completely rebuild and use a nonstandard version of it in the future has been made. Finally, medium precision means that a standard needs minor clarifications, but they could be made at a small cost and with reduced risks. When some tailoring is unavoidable due to problems in other parts of the standard, this type of standard is equivalent to a high precision one.

E. Generality

The generality criterion examines how suitable a standard is for different projects. Most standards have been designed for specific types of projects, presumably the ones that their creators were involved with; use of such standards on projects outside their original scope may cause problems. Many remarks made on the independence and precision criteria (see Sections III-B and III-D) apply here as well; the most important thing to remember, though, is that many organisations do revolve around a fixed set of projects that are adequately served by an inflexible and specialised standard. This criterion should not be confused with the independence criterion, as it applies to projects instead of methodologies.

A project has numerous attributes that could be considered as characteristic of its nature: we must first choose some of these as most relevant and then apply the criterion on them. A common classification of projects is based on their scale. As the size of a project grows, it is helpful if a standard supports division of requirements to volumes with each set of requirements communicating with other sets through a set of well-defined interfaces, possibly specified separately too. This division of requirements to internal for a module and external among modules not only eases managing large sets of requirements, but is also useful when development proceeds in builds [11], with each build delivering a part of the final product or with subcontractors implementing the separate modules. To co-ordinate the requirements, provisions for global cross-referencing should also be made, something eased by extensive use of configuration management information. In contrast, a standard designed for large projects may impose excessive bookkeeping for a small project that does not need all this information. For this aspect of generality, we can separately characterise standards according to their applicability to large and small projects. However, the characterisation of projects as large or small is difficult, so we will have to work backwards and accept a division based on whether a project needs the extra flexibility provided by multiple volumes of requirements or not. This results in the characterisation of an intuitively small application as large, if due to security reasons it must be built by separate contractors communicating through interface specifications. We nevertheless believe that this categorisation is useful, even though it is not always accurate in characterising the scale of a project, because it draws attention to this aspect of existing standards.

Another characterisation attribute is the nature of the work that the software accomplishes. Such characterisations can be even more controversial than those based on scale, but we believe that this aspect is too important to be overlooked; again, it is considered here because many existing standards were designed with specific types of applications in mind, a good example being the previous US Department of Defence standards [10], [14]. We will follow the division of projects into two categories according to these standards: embedded and automated information system development projects. Embedded systems are integrated hardware/software systems, with clear and measurable requirements, special attention to nonfunctional requirements and strict interfaces with hardware and the external environment. Automated information systems are usually software-only systems performing data processing tasks; their functional requirements are expected to change during development, as users test the system prototypes, with nonfunctional requirements having a secondary role, supporting deployment of the software to a permissive environment. Evaluation here considers the intended use of a standard, with some standards being easily applicable to both types of projects and some not. Note that even though specialisation to a specific type of project could in principle make a standard better, experience shows that it is possible to have the best of both worlds [11], [17].

As a last word here, we stress that these aspects are not the only relevant ones for any case. An organisation should select categorisation aspects that split all projects to those that are interesting for it and those that are not. By using a binary scale (applicable/not applicable) for each one of these aspects, evaluating a standard under this criterion reduces to filling in a checklist that should reveal the applicability of a standard for the circumstances. The aspects presented here are just two examples of how generality can be applied and are motivated by similar distinctions that are made in published standards and experience which shows that they are quite relevant in practice.

F. Organisation

The organisation criterion indirectly characterises the understandability and ease of use of SRSDs produced using a standard, by examining if the information in the SRSD can be presented in layers of detail, and if the outline is flexible enough to fit different situations without deviating from the core standard. Both aspects are influenced by the organisation of the proposed outline, hence the name of the criterion. Owing to the use of the SRSD in various times during development by people with different needs, it is a challenging task to organise the information contained in it so as to make it easily accessible to the user. As the SRSD will need to be complete to be an adequate basis for development it must contain a wealth of information, even though a given user would like to deal with the needed subset of the requirements only. Its authors should try to keep requirements in context, giving users an overall view of the system and the way that each requirement fits in, while at the same time helping users to locate quickly what they need. In essence, the authors should maintain clean paths from a software overview to the detailed requirements, separating different aspects of the specification in an intuitive manner.

To achieve these goals, some standards include introductory sections in the SRSD, containing overviews of the requirements, and sections describing the layout of requirements on the specific document. Requirements may be presented in layers of detail, starting from the overview and proceeding to lowest level. The organisation of the SRSD should reflect the intricacies of the specific project, especially in the way that the aspects of the requirements are tied together and in the weight placed on each category. Understandability and ease of use of the SRSD have more important roles as development, operation and maintenance proceed, because as time passes, people using the SRSD will be less familiar with the context and the internals of the software. To ease the job of these users, we consider support for layering of the requirements advantageous; to accommodate the specific needs of a project we would like the standard to provide flexibility in the SRSD structure. Note the relation of this criterion with the generality criterion presented in Section III-E: generality refers to applicability of a standard for categories of projects in a general sense, and organisation concentrates on its flexibility with respect to project-specific tailoring of the SRSD.

To develop an evaluation scale, we should first prioritise our interests among these distinct but logically related concepts. In our view, layering is a widely applicable criterion, being more important when maintenance is anticipated to be substantial, but relevant for any other project too; so it should be considered as more essential than flexibility, which may not be as important in some cases, for example when an organisation deals with fixed projects which can be accommodated by a single SRSD layout. Thus, a standard would rate as high in the organisation scale if it provides both flexibility and layering. Flexibility may be provided either by multiple supported layouts [18] or by specific provisions for tailoring [11]; to take advantage of flexibility, the SRSD should contain a section describing the organisation of the specific document at hand. The most important provision for layering is support for presenting the requirements in layers of detail. On the other end of the scale, a standard would rate as low in organisational matters, if it does not support any kind of layering or alternative organisations, but simply bundles requirements into a given set of categories; such a standard requires a lot of work from the authors to relate requirements with each other, and its maintenance can become exceedingly costly because it will be difficult to understand. Nevertheless, it can still serve as a checklist for requirements and as an easy to use guide during verification and validation, as discussed in Section III-H on view completeness. Following our prioritisation, a standard ranks medium in organisation if it supports layering, but the structure of the SRSD is static, hindering flexibility. Note that flexibility can be dangerous; multiple outlines do not seem to be a problem if their number is limited and their organisation is based on the same ideas; tailored standards on the other hand can become non-standard, and care should be taken not to stray away from the basic philosophy of the standard and the its guidelines for tailoring.

G. Content Completeness

The first aspect of completeness is the inclusion of all requirements pertaining to software development; this is examined by the content completeness criterion. Although a standard cannot enforce content completeness, it can help by including in its content outline all categories of requirements that form an adequate SRSD. This view of completeness is relevant when the standard is used as a guide for analysis and as a checklist for reviewing the SRSD. The criterion does not examine completeness in coverage of the requirements in the SRSD: a careful analysis process is the only way to include all the software requirements.

Inspection of existing standards indicates that more variety is encountered in nonfunctional requirements, where most outlines spend more time. This does not mean that functional requirements are easier to analyse and specify or less important; it is an acceptance of the fact that nonfunctional requirements are easier to overlook, as they do not relate to the main goals of the software but to peripheral, but nevertheless important, details. More elaborate categorisations of such requirements are encountered in the military standards, especially those designed for development of embedded systems. In contrast, software developed for data processing is generally more permissive, and in many cases some categories of nonfunctional requirements are completely irrelevant. An interesting relationship exists with the precision criterion (see Section III-D): whenever a standard bundles multiple categories of requirements under general headings, it is important to describe each category of requirements adequately to avoid misunderstandings; at the same time, the simple provision of many headings that cover separate categories of requirements is useless without a clear definition of what is meant by each one, because many types of nonfunctional requirements emerge under various names due to the fact that the terminology for nonfunctional requirements is not itself standardised.

An evaluation scale divides the standards in three categories. High content completeness means that a standard covers all types of requirements and in unique sections of the SRSD. These standards may seem too loaded for projects that do not require that much detail, but leaving sections blank is easy and unambiguous. On the other hand, a standard with low content completeness either completely overlooks some requirements, or it mentions them in multiple locations; such a standard may be the product of an organisation that rarely deals with such requirements, thus ignoring them or just referring to them in notes. A standard with medium content completeness makes an effort to cover all types of requirements, not suffering from many omissions, but it bundles many requirements together in a counter-intuitive manner. Tailoring here could simply consist of elaborating the outline to eliminate inconsistencies and ambiguities, although for many cases simple deletion of unused sections would be enough.

H. View Completeness

View completeness examines the ability of a standard to serve as the basis for SRSDs satisfying the needs of their diverse users. This is a second form of completeness; an external, or user-oriented, point of view, in contrast to the internal, or content-oriented, view of the content completeness criterion (see Section III-G). This is an important concern because users with different needs and backgrounds are involved with the SRSD, interacting with it throughout development and maintenance; a standard may thus try to help some users at the expense of inconveniencing others, as it is very difficult, if not impossible, to cater for everyone without overloading the document and thus hindering its maintenance. We were motivated in defining this criterion by an apparent orientation of some published standards towards specific user needs; it seems that this preference is embedded in the design and tailoring the standard to rectify it can be extremely hard, so exposing this aspect of a standard seems to be beneficial in assessing its applicability to different user populations.

To define an evaluation scale for this criterion, we must first define a categorisation of users of the SRSD. Instead of concentrating on user background, such as technical, nontechnical and managerial, we prefer to base our categorisation on the roles of the users during development; this distinction allows people to assume multiple roles. We assume that any usable SRSD will somehow cater for the needs of all of its users, or else it will not be adequate for development anyway; what we examine here is how easy a standard makes the job of specific users. We do not try to compare the standards according to how well they satisfy the needs of their users: this criterion reveals the general orientation of a standard towards some users and not how well the standard performs compared to others. Alternatively, this criterion examines how well a standard is balanced. The evaluation here will actually be just a list of the categories of users that are favoured more than others by a specific standard and it should be taken as a characterisation of a standard in isolation instead of as a comparative measure.

The users that we will consider here are restricted to those recognised as distinct parties in any set of circumstances. The software users are the sources for the requirements and the SRSD should be checked against them to verify that it is accurate; they would like it to be understandable, easy to follow and simply organised. The analysts gather requirements and write the SRSD, so they would prefer a logical and complete SRSD outline to guide them during analysis and specification. The designers use the SRSD to develop the software, and they would like a complete picture of it in adequate detail, in a document that is understandable and layered so that they receive only what they need. The testers verify and validate both the SRSD and the software against users and the SRSD, so they would like an SRSD that is easy to use for reference, is verifiable, is complete and is traceable. Finally, the maintainers would like an easily understood SRSD which is concise, detailed and easy to modify. As in the generality criterion in Section III-E, our results are based on inspection and the stated goals of standards.

I. Modifiability

The modifiability criterion examines how easy it is to modify the SRSD's contents without deterioration in its quality. Note that modifiability refers to changes in the content of the SRSD and not in changes to the outline proposed by the standard, an aspect covered by the flexibility part of the organisation criterion (see Section III-F). In practically all situations, requirements are analysed and specified in an overlapped fashion, as requirements engineers elaborate on the system level requirements. This process of analysing, recording, verifying and modifying requirements is even more pronounced on evolutionary and prototyping models, or when the project is carried out in phases in which requirements are developed separately. Such modifications may be continued during subsequent phases, and major revisions may be made even during maintenance, especially if the functionality of the software is going to be extended. We emphasise the modifiability of the SRSD during maintenance, as it is likely that at this point people dealing with requirements may have had no previous experience with the them.

We have already seen some criteria related to modifiability: organisation (see Section III-F) examines the ease of locating requirements, but not the ease with which these modifications can be made; view completeness (see Section III-H) considers the orientation of a standard towards various users, including those that make changes to it; this criterion deals only with orientation and not with adequate coverage of specific needs. To place modifiability into perspective, we must gather under it all qualities that help SRSD modifications. To ease modifications without adverse effects on the quality of the SRSD, the standard should promote a clear structure that isolates different requirements modules from each other, helping the users to find their way around the document, encourage the SRSD authors to group-related requirements together in meaningful modules that present low coupling among themselves to localise the effects of changes, and promote traceability of requirements both backwards, from software to system requirements, and forwards, from software requirements to design, implementation and testing.

An evaluation scale for the modifiability criterion would split the standards into two categories. Standards depicting adequate modifiability logically distribute the requirements into modules that demonstrate low coupling with each other, and encourage traceability through a structure that clearly shows how requirements evolve. Traceability is aided by mandatory numbering and cross-referencing of requirements through unique identifiers. Even though traceability in particular can only be encouraged by a standard, a modular structure to ease clean modifications can be enforced by the outline. Inadequate modifiability refers to all other standards; it basically means that there is no significant provision for structuring requirements into conceptual modules or explicit support for traceability information. As modifications to the requirements are the rule rather than the exception, modifiability should be an essential concern in all but the smallest, simplest and most short-lived of projects.

J. Applying the Criteria

As mentioned in Section III-A, the criteria intend to cover all aspects of a standard, to reveal as much of its nature as possible. In addition, the evaluation scales presented in the preceding sections may not be adequate for specific combinations of projects and organisations. In this Section we discuss the selection of the most appropriate criteria for any given case and their subsequent application for tailoring purposes. We consider application of all criteria using our sample scales as a very useful starting point in any case; when an organisation is selecting a standard for use in all of its projects in the future, application of all the criteria is highly recommended, because by applying more of the criteria, more details are revealed about the possible shortcomings of a standard. When a complete set of standards is adopted, the tailoring that will follow selection will be greatly eased by a thorough understanding of the situation, with the cost for this study amortised among all subsequent projects. Owing to their help with tailoring, the criteria are useful even when a standard has been externally enforced on the organisation.

A more difficult situation arises when the effort to be spent on selection and tailoring should be restricted, either due to time constraints or because the standard will be used sparsely. There, only a subset of the criteria can be used, chosen so as to reveal the most relevant characteristics of the standards under examination. We have previously given guidelines on the applicability of the criteria along with their definitions; in the following we summarise these guidelines and split the criteria into three categories depending on their general applicability. The first group of criteria is those that are relevant in practically all situations:

- precision assures that the proposed standard outline is clear and unambiguous.
- organisation examines whether the SRSD is easy to understand and use in a project.
- content completeness looks at how comprehensive the standard is in covering requirement types.

Together, these criteria examine whether a standard can be used more or less as is, without grave shortcomings. The second set of criteria are relevant in all situations, at least as viewpoints, because they show the orientation of a standard; they do not need to be used to examine the applicability of a standard for all uses, as they can simply be used to assess its performance for a specific situation:

- generality reveals the applicability of a standard for specific projects; it can also be used to examine how the standard performs under different circumstances.
- view completeness should be looked at when attention is paid to some classes of users; it is also useful for revealing the attitude towards all other users too.

The last set of criteria are completely situation-dependent, and they should be used only when the corresponding viewpoints are considered essential by the organisation;

- independence is relevant when multiple development methods or standards from other sources are used.
- integration is relevant when a complete series of standards is being adopted for all development phases.
- modifiability is relevant when SRSD changes are expected to be intense during and after development.

With this categorisation in mind, an organisation should use all criteria from the first group, some viewpoints from the criteria in the second group, and any relevant criteria from the third group.

After selecting the most appropriate criteria for the circumstances, it is advisable to prioritise the criteria, so that resolution of conflicts will be eased in the final selection phase. We do not expect a single standard to perform excellently under all of the chosen criteria; instead, we suggest that all criteria should be applied to reveal a standard's deficiencies, and performance under each criterion should be weighed appropriately to make the final decision. As this process becomes more and more complicated as the number of candidates increases, the organisation may begin its search by a preselection phase using the tables presented in Section IV to eliminate some candidates, and then apply the chosen criteria more carefully to the remaining ones; the evaluation scales defined and the evaluation of the standards presented here were designed to facilitate such a process.

To actually apply the criteria, the organisation should customise the evaluation scales. The sample scales given here are for demonstration and preselection purposes only. For any given set of circumstances, these scales may not be sufficiently fine-grained or they may put in the same category standards with differences considered important for the project. There are two ways of customising the scales: for some criteria, the performance of the standard under different circumstances is examined; there, it is essential to determine what circumstances are relevant for the organisation, and then define an evaluation scale that distinguishes between different applicability levels for these circumstances. For other criteria, the organisation may accept more or forever risks; accordingly, the evaluation scales should distinguish between standards that are satisfactory and those that are not, distinguishing between different performance levels for the former. In essence, the definition of a scale is equivalent to focusing a generic viewpoint into an adequate detail level, so that unusable standards are thrown away and usable ones are differentiated.

The last part of the process is tailoring. Application of the standards should have produced a list of problem points in the final winner; this list, along with other tailoring requirements necessary to adopt the standard to organisational practices, is the basis for an organisation-wide tailoring process that would restandardise the documents for use in all relevant projects. Tailoring should follow a set of general guidelines, so that no confusion will arise during deployment of the standard; these should be available to users of the tailored standard, so that project-specific tailoring can also take place without creating inconsistencies. A set of clear guidelines will also be helpful for people that are familiar with other variants of a standard. Many useful guidelines and ideas for tailoring can be found in the DoD Military Standard [17].

IV. Evaluation of Existing Standards

A. Scope and Purpose

In the following sections we briefly comment on published standards; our purpose is not to give an introduction to the standards, nor find the best one for all or given circumstances. We restrict ourselves in brief overviews of the standards of presenting our conclusions after applying to them our proposed criteria (see Section III). To make our results clearer, we grade the standards according to the sample scales developed during the presentation of the criteria. Our goals in this are multiple:

- Demonstrate that the criteria are relevant for an organisation, because they cover standards from enough viewpoints to be sufficient for most circumstances, whereas their use can aid in discovering the essential aspects of a standard, making an informed choice among them easier.
- Give specific examples of applying the criteria and the sample scales to real standards, to further clarify the views expressed in the preceding sections and help organisations relate our conceptual criteria to the real-life concerns that motivated their development.
- Provide a simple knowledge base for an organisation that is looking for a standard, so that it will be able to examine only the relevant ones during a customised evaluation and selection process (see Section III-J).

For all these reasons, our evaluation applies all criteria to each standard, even though it is highly unlikely that all of them would be used in any specific case. In addition, we do not place any significant value on our sample scales, other than that they are illustrative enough for our purposes as outlined above, and they can serve as a basis for preliminary selection of standards.

In the following, we look at five standards, covering more than a decade of development. They were published by the Institute of Electrical and Electronics Engineers (IEEE) [18], the European Space Agency (ESA) [20], the National Aeronautics and Space Agency (NASA) [21], [22], the Jet Propulsion Laboratory (JPL) [23] and the US Department of Defence (DoD) [11], [12], [13], [17]; the now obsolete DoD standards [10], [14], [15], [16] are also mentioned to compare approaches and illustrate their historical development. We will restrict our comments to the parts dealing with software requirements, adding only enough information to justify some evaluations and general comments that we consider important.

B. IEEE Standard

The Institute of Electrical and Electronics Engineers (IEEE) has published standards for many activities and deliverable products of the software life cycle; its 1984 SRSD standard [18] was also adopted by the American National Standards Institute. This standard has been very influential, not only because it originated from an independent organisation which explicitly attempted to accommodate as many different situations as possible, but also because its presentation is readable, concise and instructive. An outstanding characteristic of this standard is the provision for multiple SRSD outlines which structure the requirements to best serve different projects or organisations. This has generally positive repercussions on nearly all of the criteria, so it is surprising that subsequent standards have not adopted this feature.

On a more detailed level (see Table I), the standard shows medium independence as it was designed for waterfall models, stating that it cannot be used for rapid prototyping, even though it does not enforce any method of work or even the adoption of other IEEE standards; such standards exist for many, but not all, development phases, having been designed by different groups, and thus not depicting the uniformity of other series; thus medium integration applies. Its high precision is exemplary, being a consequence of informative examples rather than restrictive definitions. Concerning generality, the standard can support any type of application without problems, but it will have trouble with projects that need externally segmented requirements due to size of subcontracting, because there is no special provision for division of requirements to multiple volumes.

The multiple outlines provided, indirectly affect all the remaining criteria. They provide cheap but restricted flexibility, being uniform enough to avoid creating misunderstandings; layered specifications are supported and enhanced by the choice of the appropriate outline, so the standard shows high organisation. High content completeness applies because all types of requirements are included and the required emphasis can be placed on each one depending on the chosen outline. Each user of the SRSD will be covered adequately by any outline, with second-order trade-offs possible, so we have high view completeness. Finally, modifiability is adequate because a modular structure of requirements is supported within the SRSD, with varying coupling according to the outline chosen; traceability is eased by subdivision of requirement types inside the modules, if needed.

C. ESA Standard

The European Space Agency (ESA) published a complete series of standards [20] in 1987, dealing with both processes and

Criterion	Evaluation
Independence	Medium
Integration	Medium
Precision	High
Generality	Not for large projects
Organisation	High
Content completeness	High
View completeness	Satisfies all users
Modifiability	Adequate

TABLE I
EVALUATION OF THE IEEE STANDARD.

Criterion	Evaluation
Independence	Medium
Integration	Medium
Precision	Low
Generality	Not for large projects
Organisation	Low
Content completeness	Medium
View completeness	Satisfies only testers
Modifiability	Inadequate

TABLE II
EVALUATION OF THE ESA STANDARD.

products of the software life cycle. Unfortunately, the coverage remains at the textbook level, with the SRSD description in particular being incomplete, vague and even ambiguous at times. This is surprising considering that ESA deals with specific kinds of projects and has even adopted a life cycle model; the standard is full of advice on many aspects of development, including documentation, but is far from a practical proposal for application. Another major shortcoming of the standard is its distribution of different types of requirements in completely separate sections, in a checklist-like manner.

Concerning the criteria (see TableII), the adoption of a waterfall model has influenced the design of the standard, but the presentation is too high level to actually constrain the SRSD, so we have medium independence. Medium also applies to integration, since the series is complete but inappropriate for use as is, with intense tailoring needed. Precision is low, maybe exceedingly so, and this is the first reason for problems with the standard. Concerning generality, the standard can be used in any type of project, adopting a neutral position, but it does not support segmentation of requirements into volumes.

Organisation rates low, because the standard is inflexible, having no provisions for alternate presentation styles, and does not support any type of layering of requirements; additionally, its checklist-like outline does not seem to be appropriate for any non trivial project. This is the second reason for problems with the standard. Content completeness is medium, as most requirements are covered, but their distribution to sections is somewhat imprecise. View completeness suffers for nearly all users due to the precision problems of the standard and its organisation; however, testers of the software will probably appreciate the presentation of requirements in lists that facilitate separate checking. Finally, modifiability is inadequate since there is no support for modular requirements, with the separation of types of requirements to distinct sections, hindering safe modifications.

D. NASA Standard

The National Aeronautics and Space Administration (NASA) published its own series of standards in 1989, covering all development activities and products. It supports both monolithic and subcontracted or phased, development, either before requirements analysis or before implementation. The outlines are short and concise and they concentrate on documentation only. There are actually two standards, one for the requirements proper [21] and one for documenting interfaces [22] among pieces of the system; the latter documents are used when requirements are distributed in multiple SRSDs. As expected, nonfunctional requirements are emphasised, and configuration management and traceability information are included. An notable characteristic of the standard is its complete lack of support for modularisation within one SRSD, indicating a preference for segmenting requirements in volumes; this can impose premature design constraints on development though. Thus, interface documents will be needed even for small projects, making documentation maintenance harder.

On specific criteria (see Table III), the standard shows high independence, because it accommodates multiple life cycle models in its basic design. Integration is also high as the standard is part of a very comprehensive series; the uniformity among documents is also high. Precision is high, with clear descriptions and short examples provided. For generality, all types of applications are supported, even data-intensive ones like commercial applications; in terms of scale, small projects will suffer from excessive support information and premature design constraints due to the unavoidable segmentation of requirements.

Organisation is low: no layering is supported, and flexibility is provided only in segmenting requirements and not in organising them within a single SRSD. Content completeness is high, since all types of requirements are covered with detailed descriptions in the standards. Examining view completeness we see that the standard best serves testers and designers with its flat outline; analysts and application users will find it hard to compose and understand the SRSD due to its nonmodular structure. Finally, modifiability is inadequate, even though there is support for traceability, because there is no support for a structure that isolates requirements within modules; for this reason, maintainers are not well served by this standard.

E. JPL Standard

The Jet Propulsion Laboratory (JPL) published in 1988 its series of standards for activities and deliverables of software development. The series is based on a life cycle model that segments the system before requirements analysis. Nevertheless, the standards for the SRSD [23] can be used either with or without segmentation; the standard includes interface documents for the former case. Although JPL is associated with

Criterion	**Evaluation**
Independence	High
Integration	High
Precision	High
Generality	Not for small projects
Organisation	Low
Content completeness	High
View completeness	Satisfies designers and testers
Modifiability	Inadequate

TABLE III
EVALUATION OF THE NASA STANDARD.

Criterion	**Evaluation**
Independence	High
Integration	High
Precision	High
Generality	Suits all projects
Organisation	Medium
Content completeness	High
View completeness	Satisfies all users
Modifiability	Adequate

TABLE IV
EVALUATION OF THE JPL STANDARD.

large engineering projects, the standard is applicable to a wide variety of situations, avoiding the trap of being simply a checklist, by encouraging a modular structure. The standard is an attractive overall proposal for direct use without tailoring.

Considering the criteria (see Table IV), independence is high, despite JPL's adherence to a fixed life cycle model. Integration is also high, with a complete set of standards available that can be used, as is in most cases. The standard's precision is high, with clear descriptions supported by examples. Generality is surprisingly wide, with all types of applications catered for and both small and large projects possible; interface documents exist but the modularity provided within the requirements can eliminated them in small projects.

The organisation of the standard is medium, since layering and modularity are supported with sections describing the form of the SRSD, but there is no real provision for alternative document outlines. Content completeness is high, with many aspects that are usually overlooked being present. The insistence of the standard on traceability and configuration management information helps maintainers and testers, modularity helps users and designers and precision eases the job of analysts; thus, we have excellent behaviour in all aspects of view completeness. Last, modifiability is adequate with support for both modularity and traceability.

F. DoD Standard

The US Department of Defence has a long history in the area: until recently, DoD supported two different series of standards, the 7935A for automated information systems [10] and the 2167A for embedded systems [14], [15], [16]. At the end of 1994 the 498 series of standards replaced these documents, merging them into a single proposal that is intended to support all development projects of the DoD. For this attempt to succeed, the 498 standard should present additional flexibility without being too generalised to serve as a basis for development. To keep the standard's size within reasonable limits, the basic document [11] is supplemented by descriptions for the SRSD proper [12] and the interface requirements [13]; an additional guidebook [17] explains the design rationale of the standard and shows how it can be used, replacing the previous ones. The guidebook exposes many of the shortcomings of the previous standards and explains the solutions adopted in 498.

The standard is mainly a rework of 2167A, with added sections on environment requirements and data descriptions from 7935A. The organisation is more natural, precision has increased, examples have been given in all cases, and the documents are more uniform and less loaded with bureaucratic information. The goals stated in the guidebook, which include support for multiple models and methods, concentration on development rather than paperwork, support for emerging ideas and increased flexibility and generality, have been achieved to a quite satisfactory degree, and only minor criticisms can be made. Most of these goals are promoted by the provision of a clear strategy for tailoring the documents to best serve the needs at hand; the basic outlines are extremely comprehensive, but tailoring out sections is accepted, encouraged and supported by the documents.

Concerning the criteria (see Table V), independence is high, this being one of the design goals of 498 and a radical departure from the data oriented 7935A. The standard supports but does not enforce segmentation of the system before requirements analysis; the guidebook also shows the standard applied to different development strategies. Integration is also high, with 22 documents being specified, all of which have a uniform structure; the improved compatibility among the SRSD and the interface requirements should be evident after a comparison with 2167A. Precision is high, having been vastly improved compared to 7935A, with many examples provided in all sections. Concerning generality, the standard supports all types of projects, combining the elements of both previous proposals; for project size, the provision for modular requirements and internal interfaces makes the use of interface documents optional; thus, large projects are catered for and small projects are not impossible. Despite progress in this area though, the standard is still loaded with bureaucratic information.

Organisation is medium to high, because even though multiple layouts are not provided, substantial tailoring is supported by the standard; this is eased by the comprehensive layouts that can be cut down as required without altering the form of the document, and the inclusion of information describing it. Layering is supported at both the overview and the requirement-subrequirement levels. Content completeness is high, with all types of requirement included in a natural organisation; functional and nonfunctional requirements have been combined in the specification of each module, with elements of the descriptions coming from both previous standards. Examining view completeness we see that the standard's organisation helps all

Criterion	Evaluation
Independence	High
Integration	High
Precision	High
Generality	Not for small projects
Organisation	Medium to high
Content completeness	High
View completeness	Satisfies all users
Modifiability	Adequate

TABLE V
EVALUATION OF THE DOD STANDARD.

users of the SRSD, being natural, easy to follow, comprehensive and modifiable. Finally, modifiability is adequate, because the structure of the requirements can be modular and traceability information is explicitly included in the SRSD.

V. CONCLUSION

The proliferation of standards for software requirements specification documents is a clear indication of the importance of standardisation in this area; at the same time, the standards are so many and differ so much that an organisation considering adopting one may find itself facing a challenging task. We discussed the possible advantages and disadvantages of standardisation in Section II, concluding that it can be quite beneficial for an organisation if performed wisely.

The most probable scenario is that no one standard will exactly fit any given situation. Thus, barring the very costly solution of developing a customised standard from scratch, tailoring an existing standard will be the best approach; considering the associated risks, it is advantageous to select the standard that best fits the circumstances. Our evaluation criteria presented in Section III aim to help an organisation examine a candidate from all relevant viewpoints, revealing its applicability for any given case and pointing out its deficiencies to ease tailoring. We also discussed the process of selection and customisation of the criteria depending on the situation, as well as their application for selecting and tailoring a standard for subsequent use.

The paper included in Section IV an examination of some well known documentation standards under our proposed criteria. This generic evaluation may not be adequate for actually selecting a standard for use, but it can be quite helpful for understanding the criteria themselves, also serving as a first step towards a more focused evaluation, selection and tailoring process, that will concentrate on fewer, more appropriate standards, using customised and more detailed evaluation scales for the most important of the proposed criteria. We believe that our framework can significantly reduce the time and cost for selection of a standard, because the criteria will help an organisation concentrate on the more essential aspects, at the same time leading to the selection of the best standard for the circumstances and easing considerably the final tailoring process.

REFERENCES

[1] I. Sommerville, *Software engineering*, Addison-Wesley, Reading, MA, 4th edition, 1992.

[2] B.W. Boehm, *Software life cycle factors*, in C. Vick and C.V. Ramamoorthy (Eds.), *Handbook of software engineering*, Van Nostrand Reinhold, New York, NY, 1984.

[3] R.H. Thayer and M.C. Thayer, *Glossary*, in M. Dorfman and R.H. Thayer (Eds.), *Standards, guidelines, and examples on system and software requirements engineering*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[4] M. Jazayeri, C. Ghezzi and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[5] J.W. Bracket, *Software requirements, SEI curriculum module*, SEI-CM-19-1.2, Carnegie Mellon University, Software Engineering Institute, 1990.

[6] R.S. Pressman, *Software engineering, a practitioner's approach*, McGraw-Hill, 3rd edition, 1992.

[7] M. Dorfman and R.H. Thayer (Eds.), *System and software requirements engineering*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[8] A.M. Davis, *Software requirements, analysis and specification*, Prentice-Hall, Englewood Cliffs, NJ, 1990,

[9] M. Dorfman and R.H. Thayer, *Introduction to tutorial*, in M. Dorfman and R.H. Thayer (Eds.), *Standards, guidelines, and examples on system and software requirements engineering*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[10] *System/subsystem specification*, DoD Military Standard DOD-STD-7935A, US Department of Defence, 1988.

[11] *Software development and documentation*, DoD Military Standard MIL-STD-498, US Department of Defence, Washington, DC, 1994.

[12] *Software requirements specification DID*, DoD Military Standard DI-IPSC-81433, US Department of Defence, Washington, DC, 1995.

[13] *Interface requirements specification DID*, DoD Military Standard DI-IPSC-81434, US Department of Defence, Washington, DC, 1995.

[14] *Defence system software development*, DoD Military Standard DOD-STD-2167A, US Department of Defence, Washington, DC, 1988.

[15] *Software requirements specification DID*, DoD Military Standard DI-MCCR-80025A, US Department of Defence, Washington, DC, 1988

[16] *Interface requirements specification DID*, DoD Military Standard DI-MCCR-80026A, US Department of Defence, Washington, DC, 1988.

[17] *Draft guidebook for MIL-STD-498, overview and tailoring*, DoD Military Standard, US Department of Defence, Washington, DC, 1995.

[18] *IEEE guide to software requirements specifications*, ANSI/IEEE Std 830-1984, The Institute of Electrical and Electronics Engineers, New York, NY, 1984.

[19] R. Balzer and N. Goldman, *Principles of good specification and their implications for specification languages*, in N. Gehani and A. McGetrick (Eds.), *Software specification techniques*, Addison-Wesley, Reading, MA, 1986.

[20] *ESA software engineering standards*, ESA PSS-05-0, Issue 1, European Space Agency, ESA Publications Division, ESTEC, Noordwijk, The Netherlands, 1987.

[21] *Software requirements DID*, SMAP-DID-P200-SW, Release 4.3, National Aeronautics and Space Agency, 1989.

[22] *External interface requirements DID*, SMAP-DID-P210, Release 4.3, National Aeronautics and Space Agency, 1989.

[23] T.J. Fouser, *Software requirements analysis phase*, JPL D-4005, Version 3.0, Jet Propulsion Laboratory, Pasadena, CA, 1989.