*Reliable Computing* (2007)
DOI: 10.1007/s11155-006-9020-7


# Requirements Analysis for Engineering Computation: A Systematic Approach for Improving Reliability

SPENCER SMITH, LEI LAI, and RIDHA KHEDRI
*McMaster University, Computing and Software Department, 1280 Main Street West, Hamilton, Ontario, L8S 4K1, Canada, e-mail: smiths@mcmaster.ca*

(Received: 22 October 2004; accepted: 17 May 2005)

**Abstract.** This paper argues that the reliability of engineering computation can be significantly improved by adopting software engineering methodologies for requirements analysis and specification. The argument centers around the fact that the only way to judge the reliability of a system is by comparison to a specification of the requirements. This paper also points to methods for documenting the requirements. In particular, a requirements template is proposed for specifying engineering computation software. To make the mathematical specification easily understandable by all stakeholders, the requirements documentation employs the technique of using tabular expressions. To clarify the presentation, this paper includes a case study of the documentation for a system for analyzing statically determinant beams.

## 1. Introduction

Software engineers generally advocate that the first step in system development should be a systematic elicitation, analysis and documentation of the requirements, because it is much easier and cheaper to correct mistakes and misconceptions at the beginning of the process than it is to try and fix problems during implementation and maintenance. There is wide agreement in the software engineering community on the necessity of a complete and consistent software requirements document for evaluating any software system quality, including reliability [4]. Requirements documentation has been demonstrated to be effective in other application areas, such as with business applications [29] and for real-time systems, such as the U.S. Navy's A-7E military aircraft [15] and the shutdown systems of the Darlington nuclear generating station [27]. However the requirements stage of software development is often neglected when solving engineering computation problems, where engineering computation is defined as the use of computer tools to analyze or simulate mathematical models of real world systems of engineering or scientific importance so that we can better understand and predict the system's behaviour. This paper argues that the reliability of engineering computation can be significantly improved by adopting software engineering methodologies for requirements analysis and specification.

The importance of requirements analysis and documentation is not only widely recognized by software engineers, all engineering disciplines understand the importance of documenting requirements for large and complex systems. For instance, an automobile manufacturer gathers requirements from customers to determine whether fuel efficiency is considered more important than luxury, or vise versa. Similarly, a structural engineer would not start designing a building until she had determined the following: How many floors will the building have? What are the expected loads? Will the building be used as a hospital, a school, a shopping mall, or for some other purpose? To judge the success of the design, it is necessary to take the requirements into account. For instance, the required reliability for a hospital is higher than for other structures, so the probability of a hospital collapsing due to an earthquake should be less than that for an office building in the same situation. Given that engineers recognize the importance of requirements for large and complex systems, and that the size and complexity of software systems, including engineering computation systems, seem to be continually growing, it is necessary to ensure the quality of software systems via documenting their requirements.

The central argument of this paper is that a requirements specification document is necessary to judge the reliability of engineering computation software. Reliability is a measure of the dependability of a system. One definition says that "reliability of software is defined to be the ability of the software to behave consistently in a user-acceptable manner when subjected to an environment in which it was intended to be used" [5, p. 310]. Another definition say that "reliability requirements deal with failures to provide service. They determine the maximum allowed software system failure rate, and can refer to the entire system or to one or more of its separate functions" [10, p. 39]. The definitions of reliability depend on the existence of a specification of the requirements because one cannot judge correctness, user acceptability, or failure rates, without knowing the standard for comparison. In engineering computation, one needs to know exactly what problem the system is required to solve and the values for the acceptable tolerances, or it is impossible to judge whether the results are correct. Although the field of engineering computation has developed many excellent methodologies for producing efficient and accurate numerical results, the design decision for selecting the appropriate methodology is often made in an ad hoc manner because there is a lack of appropriately documented requirements to guide the decision. The existence of a complete and consistent requirements document can lead to better decisions for improving reliability, and it can also improve other software qualities, such as usability, verifiability, maintainability, reusability and portability, which are sometimes neglected in engineering software.

The first section below presents background information, including an overview of software engineering methodologies for requirements elicitation, analysis and documentation. The background section also contains a brief summary of the syntax and semantics of tabular expressions, which are introduced in this document because they provide a relatively easy way of documenting complex requirements. After

the background section, the value of requirements documentation for engineering computation is explored in depth. Following this, the methodology promoted in this paper is made more concrete by presenting some excerpts from a requirements document for a software system to analyze statically determinate beams. The discussion of this example highlights the requirements template that was followed in constructing the requirements document. The final section consists of concluding remarks.

## 2. Background

The idea of this paper is to borrow guidelines from successful software engineering projects with the goal of improving engineering computation. To do this, it is first necessary to understand some aspects of software engineering. This section provides necessary information on some current software engineering methodologies. In this paper the term methodology refers to a combination of methods and techniques, where methods are general guidelines governing an activity, and techniques are the technical and mechanical approaches employed to support the methods. This background section provides an overview of requirements elicitation, analysis and documentation, and it describes a particular approach for documenting formal requirements: the technique of tabular notation. More detail on the software engineering methodologies discussed here can be found in [21].

### 2.1. Overview of Requirements Elicitation, Analysis and Documentation

This section highlights how requirements fit into the software development process by first providing a description of the waterfall model of software development. Following this, the software requirements activities are described for elicitation, analysis, documentation, validation and verification of requirements. A section is also provided to describe the end-product of the requirements phase, the document called the Software Requirements Specification (SRS). The final section describes a requirements template, which is a documentation approach used by software engineers to improve the quality of the SRS.

#### 2.1.1. *Waterfall Model*

In one model of the software development lifecycle, the first phase involves gathering requirements, analyzing them and documenting them. This lifecycle model, which is graphically depicted in Figure 1, is termed the waterfall model because each stage flows into the next as the process moves downstream. The stages of the waterfall model consist of requirements, design and coding. The back and forth arrows represent the iterative process of validation and derivation at each phase before proceeding to the next. After the code is developed and validated against the design, it also has to be validated against the requirements. This is represented

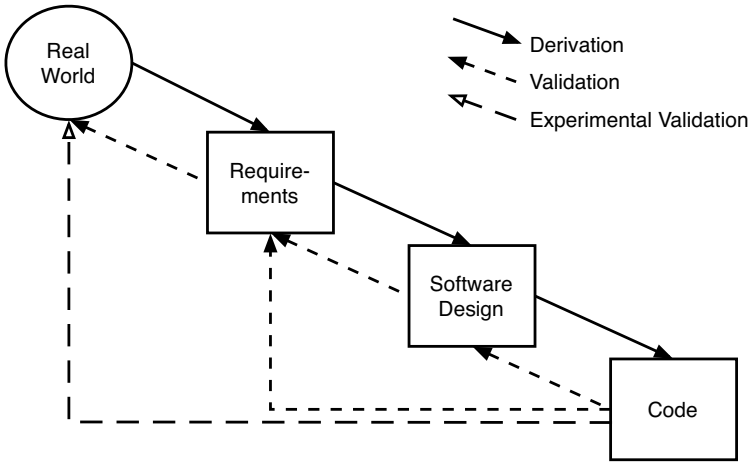


*Figure 1.* Waterfall model of the software development lifecycle.

by the dashed arrow from "Code" to "Requirements." The acceptance testing, in which the system is used in the real world, is performed before the software is finally accepted. This is indicated by the dashed arrow from "Code" to "Real World." If the requirements are modified, the whole procedure has to be repeated.

The waterfall model is not the only model of the software development process [12, pp. 385–456], but it is the model that is cited here because it closely parallels how engineers typically think about their work flow. Moreover, as Section 3 will show, the software lifecycle model and the scientific method essentially follow the same waterfall process. The waterfall model is also well-suited for engineering computation problems because the waterfall model works well when the requirements are stable [12, p. 409], which is certainly the case in engineering computation where the scientific theories of the laws of physics are slow to change. Another argument in favor of presenting the waterfall model is that even though the process of software development is never as rational as that presented in Figure 1, the advantages of a rational process can still be obtained by documenting the work products as if they were developed and written following the waterfall model [26].

### 2.1.2. *Software Requirements Activities*

To develop engineering computation software in a rational way, it is necessary to document the software requirements. A software requirement is a description of how the system should behave, or of a system property or attribute [32]. In [33], a software requirement is defined as 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.

Software requirements activities cover all of the activities involved in discovering, analyzing, documenting, and maintaining a set of requirements for a computer-based system, with an emphasis on using systematic and repeatable techniques [30], [32]. The software requirements activities begin with the software requirements elicitation. At this stage an attempt is made to work with the stakeholders to gather all of the information necessary for understanding the problem. After all of the requirements have been gathered, they are analyzed, which involves refining and modeling the requirements. The goal of analysis is to discover problems, incompleteness and inconsistencies in the elicited requirements. The analysis is interleaved with the requirements elicitation phase. Some methodologies for requirements analysis include structured analysis, object-oriented analysis [22], goal based methods [22], [35], viewpoint methods [6] and component requirements analysis [13].

For the results of the requirements analysis to be useful for the subsequent development of the software, the requirements need to be documented in the software requirements specification document, which is discussed in the next section. To ensure quality, the software requirements must undergo a process of validation and verification to check the adequacy of the documented requirements. In the validation, the requirements model is examined to make sure that stakeholders' expectations are correctly captured. Verification involves checking the software requirements for certain properties, such as consistency, completeness and modifiability.

In this paper, the term requirements analysis does double duty. It refers to the stage of requirements refinement and modeling described above, but it is also used as a shorthand to describe all of the requirements steps, from elicitation to verification and validation. In the literature, the term requirements engineering is sometimes used for this purpose. The meaning of the term requirements analysis should be clear from the context where the term is used.

In terms of quality, requirements should be reviewed to ensure that they are correct, unambiguous, complete, consistent, modifiable, verifiable and traceable [17]. A good requirement should express "What" functionalities and qualities the system should have, but it should not mention "How" these requirements are to be accomplished. That is, the requirements should not impose design decisions. For instance, a requirement may specify an ordinary differential equation that must be solved, but it should not mention that a fourth order Runge-Kutta method should be employed. The requirements document should not tie the hands of the designer; she should be free to select any algorithm that will satisfy the requirements.

### 2.1.3. *Software Requirements Specification (SRS)*

During the process of requirement gathering, the requirements need to be documented in a software requirements specification (SRS), which includes the external behavior of the system, the constraints placed on the implementation, the forethought about the lifecycle of the system and the acceptable response to the unde-

sired events [15]. The SRS is a document that clearly and precisely describes each of the essential requirements (functions, performance, constraints and quality attributes) of the software and external interfaces [33].

Requirements documentation methods can be categorized according to their degree of formality, where formality is defined as the degree to which use is made of mathematical techniques and notations. The first group of methods are informal methods, which describe the requirements document in natural language. In principle, the requirements in natural language are universally understandable, but in practice, the meaning of requirements is not always obvious, because natural language is inherently ambiguous, and analyzing such descriptions can be very difficult [19]. The second group of documentation methods are formal methods, which use mathematically formal syntax and semantics to specify system function and behavior. Example languages currently used in formal specifications are Z, VDM, CSP, etc. Formal methods do not have the ambiguity of natural language, but they can be time consuming to produce, and formal methods do not help us "solve the difficulties caused by lack of understanding of the real world situation" [2]. The last group of methods are semi-formal methods that use diagrammatic modeling or object-oriented techniques. These methods are generally easier to develop and understand than formal methods. However, semi-formal methods are facing criticism for paying less attention to verification and validation of requirements [30]. At this time, requirements documentation methods are not well-developed, and there is no universally accepted way of documenting requirements.

As the official statement of the system requirements for customers, end-users and software developers, the SRS provides many advantages during the lifecycle of the software project [17], [30], [32]. For instance, the SRS reflects the mutual understanding of the problem to be solved between the requirements analyst and the client, and the SRS serves as a starting point for the software design phase because decisions are made explicitly before designing and coding. The SRS provides a basis for estimating costs and schedules, and it allows validation and verification because it establishes a baseline against which compliance can be measured. The SRS aids the software lifecyle because it facilitates incremental development. In many businesses, systems are built in increments; that is, the next generation inherits the features from the previous version, only enhancing the system with additional or improved features. The final benefit of an SRS is the financial benefit of finding problems early. If mistakes are found in the requirements stage, then they are much cheaper to fix than when they are found in a later stage of the software development. Empirical studies show that if one arbitrarily assigns unit cost to the effort required to detect and repair an error during the coding stage, then the cost to detect and repair an error during the requirements stage is between a fifth and a tenth as much, and the cost to detect and repair an error during maintenance is twenty times as much [5, p. 25].

### 2.1.4. *Requirements Template*

A requirements template provides a frame of reference, identifies needed information and suggests an order of presentation so that the requirements can be best expressed [30]. The use of a template encourages a systematic procedure of requirements documentation. Since no single template can meet the needs of every requirements document, it is vital that the template be tailored to the needs of a particular audience [30].

The advantages of using requirements templates are discussed in [30], [32]. One advantage is that templates can increase the productivity of SRS's. Software can be developed to support the process of producing requirements documents conforming to the template. Furthermore, templates can increase the adequacy of SRS's because a well-organized format for the document acts as a checklist for writers of the SRS and reduces the chances of omitting information. Another benefit of a template is that it facilitates the communications among various SRS users, such as customers, developers, experts, etc., which in the context of engineering computation, will be researchers, software developers, physical modelers, computational scientists, etc. Templates also provide the advantage of easing information handling by defining the content of each specific section. The readers can find information more easily and understand the relationships between different parts of the document. Finally, a template helps the process of software development by making it easier to compare two SRS's when they both conform to the same template.

There are several requirements specification frameworks that are designed for general purposes and contain good advice on how to write requirements and how to avoid problems [7], [9], [23], [24], [29], [33]. These templates are the result of many years of practice, consulting and research in requirement activities and thus provide a good foundation for software requirements documentation. They are subject to change and are usually not used without modification. The templates that have been developed to date focus on business applications and real-time systems and do not address some of the issues of importance for engineering computation problems, which motivates the development of the new template discussed in Section 4.

## 2.2. TABULAR EXPRESSIONS

Tables are used for documenting the requirements in this paper because they improve readability so that formal documentation can be advocated to replace conventional documentation. Tabular expressions (or tabular notations) for computer programs and modules made their appearance in the late 1950s [18]. Multi-dimensional tabular expressions make it easier to consider every case separately while writing or reading a document, as opposed to the standard linear mathematical notation. The key ideas of tabular expressions, one of the cornerstones of the relational model for documenting the intended behavior of programs [6], [18], [25], were first developed in work for the US Navy and applied to the A-7E aircraft [16]. In the current case study, tabular expressions are used in the SRS for the beam

Table 1. System Response to Constraints on Input Variable $x_1$.

| Composition rule | $\bigcup_{i=1}^{4} H_2[i] \cap \left( \bigcap_{j=1}^{2} H_1[j]; G[i,j] \right)$ | |
|---|---|---|
| | | $H_1$ |
| | $S'_{\text{GET}} \cup =$ | $ErrorMsg' +=$ |
| $x_1 < 0$ | $\emptyset$ | $InvalidInput\_x_1$ |
| $0 \leq x_1 < \min_d$ | $\emptyset$ | $x_1\_TooSmall$ |
| $x_1 > \max_d$ | $\emptyset$ | $x_1\_TooLarge$ |
| $\min_d \leq x_1 \leq \max_d$ | $\{@x_1\}$ | $NULL$ |
| | $\wedge ChangeOnly(S_{\text{GET}}, ErrorMsg)$ | |
| $H_2$ | $G$ | |

analysis problem to formalize the specification of the system behavior. The advantages of tabular expressions are that they are well-structured, they can simplify the task of composition of table specifications to have a global or a dynamic view of the system's behavior and they facilitate the improvement of SRS quality attributes such as completeness and consistency.

A full review of tabular expressions is beyond the scope of this paper; details on the mathematics of tables can be found in [21]. An intuitive understanding of tables can be obtained by considering an illustrative example. The example is taken from the SRS for beam analysis. The SRS uses a table to specify the system response to input data for describing the beam problem and the constraints on this data. The example in Table 1 is for input of the distance from the left end of a beam to the point of application of a load ($x_1$). (Details of the beam problem and variables such as $x_1$ are presented in Section 4 and in Figure 3.) In this table $\min_d$ and $\max_d$ are the bounds on the admissible range of values for $x_1$, $@x_1$ is the symbol that represents $x_1$, $S_{\text{GET}}$ is the set of symbols of user interface variables (the variables that will either be input by the user or output by the software), *ErrorMsg* is a system output indicating the error mode, $ChangeOnly(var1, var2, \ldots)$ indicates that only the output variables $var1, var2, \ldots$ may change value, $i$ and $j$ are the indexes of the table cells, $H_1$ and $H_2$ are headers that consist of an indexed set of cells, $G$ is a grid and the composition rule is a relation expression that determines the relation represented by the table.

Although the mathematical underpinnings of tabular expressions can be complex, the interpretation is natural and intuitive. In this example the value of $x_1$ determines the new values (indicated by a prime ($'$) symbol on a variable's name) of $S_{\text{GET}}$ and *ErrorMsg*. For a given value of $x_1$ one should search for the matching predicate in $H_2$ and read the row in $G$ to determine what happens to the other variables. For instance, if $\min_d \leq x_1 \leq \max_d$, then $S'_{\text{GET}} = S_{\text{GET}} \cup \{@x_1\}$ and $ErrorMsg' = ErrorMsg$. Formally, the composition rule provides a relational interpretation of Table 1 by allowing the construction of either a homoge-

neous or heterogeneous relation from the cells of the table. For instance, if we denote the empty string by *NULL*, string concatenation by +, and we assume that $x_1, S_{\text{GET}},$ and *ErrorMsg* are respectively of type $A, B$ and $C$, then the homogeneous relation $R \subset (A \times B \times C) \times (A \times B \times C)$ associated with Table 1 is the following:

$$
\begin{aligned}
R = \Big\{ & \big((x, S_{\text{GET}}, \mathit{ErrorMsg}), (x', S'_{\text{GET}}, \mathit{ErrorMsg}')\big) \mid \\
& x_1 < 0 \wedge S'_{\text{GET}} = S_{\text{GET}} \cup \emptyset \\
& \qquad \wedge \mathit{ErrorMsg}' = \mathit{ErrorMsg} + \mathit{InvalidInput\_}x_1 \ \wedge \ x'_1 = x_1 \\
& \vee\ 0 \leq x_1 < \min_d \ \wedge S'_{\text{GET}} = S_{\text{GET}} \cup \emptyset \\
& \qquad \wedge \mathit{ErrorMsg}' = \mathit{ErrorMsg} + x_1\mathit{\_TooSmall} \ \wedge \ x'_1 = x_1 \\
& \vee\ x_1 > \max_d \ \wedge S'_{\text{GET}} = S_{\text{GET}} \cup \emptyset \\
& \qquad \wedge \mathit{ErrorMsg}' = \mathit{ErrorMsg} + x_1\mathit{\_TooLarge} \ \wedge \ x'_1 = x_1 \\
& \vee\ \min_d \leq x_1 \leq \max_d \ \wedge S'_{\text{GET}} = S_{\text{GET}} \cup \{@x_1\} \\
& \qquad \wedge \mathit{ErrorMsg}' = \mathit{ErrorMsg} + \mathit{NULL} \ \wedge \ x'_1 = x_1 \Big\}.
\end{aligned}
$$

## 3. Why Requirements Analysis for Engineering Computation?

Attempts to apply software engineering methodologies to engineering computation have usually payed little attention to the appropriate and rigorous documentation of requirements. Some ideas from software engineering have been applied to engineering computation, such as algebraic abstractions [1], object-oriented design [14], software components [8] and software patterns [3]. Although these approaches have advantages, the research usually focuses on the design and implementation and does not address how to improve the quality of engineering software from the requirements level. One exception to neglecting the requirements phase is a requirements analysis of data parallel applications [11]. Another exception documents the requirements of models of physical phenomena [20] using tabular expressions. However, this model of physical phenomena does not necessarily solve all of the problems for documenting the requirements of engineering software because the original idea of this model was developed for an embedded system, which has different needs than an engineering computation system. Moreover, this model allows the numerical methods, which are essentially implementation decisions, to be encompassed into the requirements documentation. This contradicts with the principle that requirements should not address "How," but only "What."

The apparent absence of studies on the requirements for engineering computation is not an indication that this is an unimportant topic. All of the benefits listed in Section 2.1.3 apply to engineering software, just as they do to other types of applications. Besides these arguments, arguments specific to engineering software also can be made.

The argument that requirements analysis can improve the reliability of engineering computation is made first by observing the strong similarity between the waterfall model of the software lifecycle and the standard model of the scientific

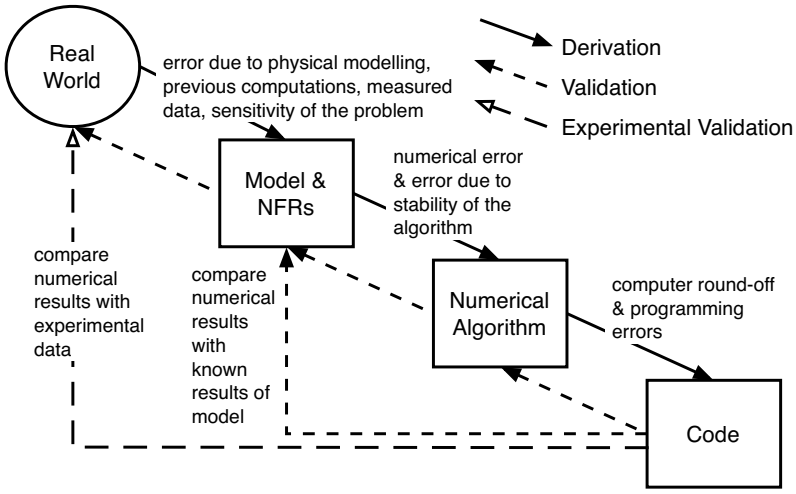


*Figure 2*. The Scientific Method.

method. The strong similarity implies that engineering computation can benefit from methods that have proved to be successful in software engineering. Like the waterfall model, the typical work flow for the development of engineering modeling and simulation software can also be divided into several stages [31], as illustrated in Figure 2. First, physical modeling is used to do the basic theoretical research, using assumptions to simplify the real world so that mathematical models can be constructed. The correctness of these models is validated against the original problem. Second the numerical algorithms are developed using a combination of scientific computing knowledge and computer science. Again, the algorithms are tested against the mathematical models, and the code is validated against the algorithms. There are two stages for correctness confirmation. First, the code should be validated against the mathematical model, which makes sure the implementation is a correct reflection of the model. Second, experiments are used to validate that the model embodied in the requirements is adequate for the intended use. If problems appear during the experimental validation, this work flow recycles to the physical modeling stage, and the same cycle is applied for the changes in the mathematical models.

In the above discussion, the terms validation and verification are used in the same sense that they are used in software engineering literature, which is different than the definitions sometimes used in the engineering computation literature. For instance, Roache [28, pp. 19–36] reserves the word validation for experimental validation, but in the current paper validation refers to the external validation of any phase in the software development process, which would be called verification by Roache. In the current paper the term verification refers to checking the internal properties, such as consistency and completeness, at each stage of the software

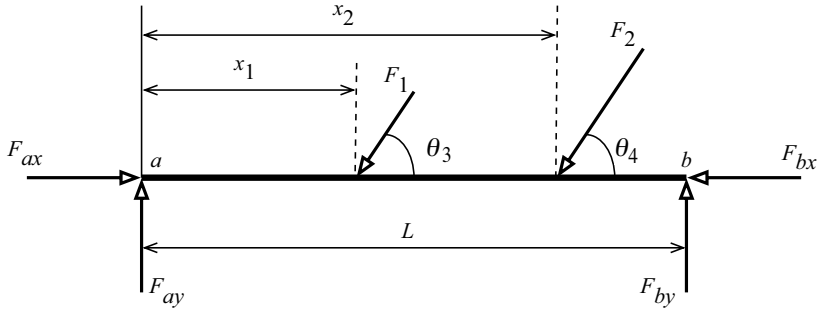


*Figure 3.* Free body diagram for beam with pin-pin supports.

development process, and the "validation" of Roache's type is identified by the compound term "experimental validation."

There exists intrinsic similarity between the processes in Figures 1 and 2; that is, each box and arrow in Figure 2 has a counterpart in Figure 1. This similarity motivates applying the innovative ideas from the discipline of software engineering to the field of engineering computation.

As stated in the introduction, another argument in favor of requirements documentation for engineering computation problems is that reliability can only be judged by comparison to explicitly stated requirements. Verification and validation (V&V) are difficult if it is unclear what standards the system is being verified and validated against. Clear requirements are necessary to know what the system should be inspected and tested for. Moreover, current V&V efforts focus on functional requirements, but the nonfunctional requirements, such as accuracy, efficiency, portability etc. are as important and should also be tested.

## 4. SRS for Beam Analysis Software

The methodology for documenting the requirements for engineering computing problems is illustrated by the example of a system for analyzing beams, called BAS (Beam Analysis Software). The central object in BAS is a beam with two external forces and supports at ends $a$ and $b$, as presented in Figure 3. The beam, of length $L$, is in static equilibrium. The following information about the beam is given: the beam properties and the external loading ($F_1$ and $F_2$ located at positions and angles $x_1$, $x_2$, $\theta_3$, and $\theta_4$, respectively). The purpose of BAS is to calculate the unknown applied forces or support reactions ($F_{ax}$, $F_{ay}$, $F_{bx}$, and $F_{by}$), the internal shear forces and bending moments, and the deflection of the beam. This software will be used as an educational tool for teaching statics and strength of material. Although the example is relatively simple there is enough complexity to illustrate the value of requirements documentation. The advantages discussed here should become even more pronounced as the complexity and size of the engineering computation problem grows.

The SRS template was constructed by borrowing ideas from the templates presented in Section 2.1.4, especially the IEEE Standard 830-1998 [17] and the Volere Requirements Specification Template [29], but it was also necessary to add original ideas to the new template. In some ways the new template is simpler than its predecessors. For instance, only one viewpoint needs to be considered for engineering software, but many different viewpoints need to be considered for business applications (e.g. the accounting viewpoint, the marketing viewpoint, etc.). Although different in the specific details, all engineering software can be abstracted as the sequence: accept input information, perform calculations and report the results. In business applications, on the other hand, the interaction of the system with the environment is typically more complex. Although the engineering computation SRS template is simpler in terms of the number of viewpoints than some templates, in other ways it is more complex. Certain issues are more important for an engineering computation domain, such as the physics of the natural world, the sensitivity of the system of equations, the accuracy of the calculations, etc. One significant change from the existing templates is for the functional requirements, which are split into two main sections: *i*) "Problem Description," which consists of a background overview, terminology definitions, physical system descriptions and goal statements; and, *ii*) "Solution Characteristics Specification," which documents assumptions, the theoretical models, data definitions, instanced models, data constraints and system behaviors. The template also introduces a new traceability matrix to facilitate future modifications to the functional requirements. Another change that benefits scientific software is tailoring the non-functional requirements so that the following are explicitly addressed: the accuracy of the input data, the sensitivity of the model, the tolerance of the solution and the solution validation strategies. The details of the SRS template, along with a discussion of the benefits it provides, are examined in the subsections that follow. Each subsection discusses a different benefit of requirements documentation and then presents excerpts from the full SRS [21] to illustrate how the benefit is achieved.

### 4.1. The Template Provides Guidelines

The proposed requirements template supports and encourages a systematic process because it decomposes the problem into smaller steps and thus provides guidelines and a checklist for the issues that need to be addressed and the questions that need to be asked. The sections of the proposed template are shown in Figure 4. The sections encourage a systematic process by forcing the authors to consider each heading, even if the eventual decision is that the heading is inappropriate for a given problem. The division into sections, which is an example of applying the principle of separation of concerns, is an engineering approach to handling large and complex problems and for facilitating multidisciplinary collaboration. One example of separation of concerns is that the organization and purpose of the document (Sections 2.a and 2.c of the SRS template) are discussed separately from

1. Reference Material: a) Table of Symbols, b) Abbreviations and Acronyms
2. Introduction: a) Purpose of the Document, b) Scope of the Software Product, c) Organization of the Document
3. General System Description: a) System Context, b) User Characteristics, c) System Constraints
4. Specific System Description:
   a) Problem Description: i) Background Overview, ii) Terminology Definition, iii) Physical System Description, iv) Goal Statements
   b) Solution specification: i) Assumptions, ii) Theoretical Models, iii) Data Definitions, iv) Instanced Models, v) Data Constraints, vi) System Behavior
   c) Non-functional Requirements: i) Accuracy of Input Data, ii) Sensitivity of Model, iii) Tolerance of Solution, iv) Solution Validation Strategies, v) Look and Feel Requirements, vi) Usability Requirements, vii) Performance Requirements, viii) Maintainability Requirements, ix) Portability Requirements, x) Security Requirements
5. Traceability Matrix
6. List of Possible Changes in the Requirements
7. Values of Auxiliary Constants

*Figure 4.* Table of Contents of the SRS for BAS.

one another and from the rest of the document so that the discussion about the documentation itself will not complicate the discussion of the requirements.

Another example of separation of concerns is that the presentation of the functional requirements (Sections 4.a and 4.b of the SRS template) is separated from the presentation of the non-functional requirements (Section 4.c of the SRS template). By separating functional (system behaviors) and non-functional requirements (overall system qualities) the analysis can focus on what the system is intended to do separately from thinking about what qualities the system should have. As an example, in the SRS for BAS, the decision that BAS solves for unknown forces is separated from the requirement that, "The maximum response time of any interaction between the user and BAS should be less than 1 second." The requirement to solve for unknown forces may also exist in another systems, but in that other system the response time may be required to be faster than 1 second. Due to the separation of the two requirements, it would be straightforward to adapt the current documentation to reflect the needs of the new system. The clear separation of these two requirements improves the potential reusability of the documentation.

Another feature that improves reusability and makes the document easier to maintain is the use of labels for cross-referencing. Whenever necessary the different parts of the SRS have their own label. The items that have labels include the following: the different sections of the SRS, the physical system descriptions (in SRS Section 4.a.iii), the goal statements (in SRS Section 4.a.iv), the assumptions (in SRS Section 4.b.i), the theoretical models (in SRS Section 4.b.ii), the instanced models of BAS (in SRS Section 4.b.iv) and all of the tables and figures used in the

SRS. As an example, physical system description PS1.a. says, "the shape of the beam is long and thin" and PS1.d says, "The transverse cross-section of the beam is rectangular." The use of labels allows the interrelationships between items in the SRS to be explicitly documented. Moreover, if there is a change in one portion of the document, it should be possible to determine what other portions of the document are affected, which improves reusability.

An additional feature of the proposed template that encourages a systematic approach and results in reusable documentation is the use of parameters instead of explicit values. Rather than say that the allowed range for $x_1$ is $0 \leq x_1 \leq 999999$, the range is written as $\min_d \leq x_1 \leq \max_d$. The actual values for these parameters are supplied in SRS Section 7, "Values of Auxiliary Constants." This approach has the same advantages as using symbolic constants in a computer program: the value is easy to change and the symbolic name for the constant improves the documentation by being more meaningful than a number. Moreover, the use of parameters in this way encourages a systematic procedure. The discussion of what value is appropriate for a constant can be separated from the decision that a variable should have some, as yet undetermined, limits on its value.

### 4.2. A Transition from General to Specific

The requirements template supports a systematic process by gradually taking the authors from general (abstract) to specific (concrete) concepts. For instance, as Figure 4 shows, the structure of the document proceeds from a general introduction to a specific system description. The "Introduction" section provides an overview of the entire SRS. After this the "General System Description" provides general information about the system, identifies the interfaces between the system and its environment, describes the user characteristics and system constraints. The next section, "Specific System Description," increases the level of detail and presents more concrete information. This section provides the physical system description, defines the system goals, presents a mathematical model of the system and documents the non-functional requirements. The structure of the template helps the SRS authors by allowing them to document the "big-picture" before thinking about the details.

The transition from general to specific also occurs with the refinement of the abstract system goals to a theoretical model and finally to a concrete instanced model of the system. The BAS system goals state that given the beam properties and some of the external forces, BAS should solve for:

**G1**. The unknown external forces applied to the beam.

**G2**. The functions of shear force and bending moment in the beam.

**G3**. The function of deflection along the beam.

These goals are refined further via a theoretical model that helps the reader to develop an understanding of the solution by introducing the theory and principles

relevant to the problem. For instance, the goal **G1** is refined by the theoretical model for equilibrium, **T1**:

$$(\mathbf{T1}) \quad \begin{cases} \sum F_{xi} = 0, \\ \sum F_{yi} = 0, \\ \sum M_i = 0, \end{cases}$$

where the forces and moments are all represented by their signed magnitude symbols: $F_{xi}$ represents the $i$th force component in the **x** direction, $F_{yi}$ represents the $i$th force component in the **y** direction, $M_i$ represents the $i$th moment component in the **z** direction. Similarly there are theoretical models **T2** and **T3**, which are detailed in [21], to refine the other two system goals.

To reinforce the fact that **G1** is more abstract than **T1**, it is worth mentioning that **T1** is not the only option for solving for the unknown forces. A theoretical model could be constructed that used the principle of virtual work, instead of using the equations of equilibrium.

The theoretical model alone does not provide enough information to solve for the unknown forces. It is necessary to introduce and define a more detailed model of the beam and the forces. In this case study the forces are represented as in Figure 3 and defined in SRS subsection "Data Definitions," with some of the forces decomposed into **x** and **y** components and others given as a magnitude and a direction. Another SRS subsection "Instanced Models" applies the theoretical models to the physical system. For instance, when moments are taken about point $a$, one instance of the model **M1** is:

$$(\mathbf{M1}) \quad \begin{cases} F_{ax} - F_1 \cdot \cos\theta_3 - F_2 \cdot \cos\theta_4 - F_{bx} = 0, \\ F_{ay} - F_1 \cdot \sin\theta_3 - F_2 \cdot \sin\theta_4 + F_{by} = 0, \\ -F_1 \cdot x_1 \sin\theta_3 - F_2 \cdot x_2 \sin\theta_4 + F_{by} \cdot L = 0. \end{cases}$$

As was the case for the transition from **G1** to **T1**, there are other concrete options available for specifying **M1**. One of the advantages of the proposed template is that it supports reuse by allowing a new theoretical or instanced model to be introduced. Any of the documentation associated with the more abstract model or goal can remain unchanged in the new version of the SRS.

### 4.3. Special Cases Are Considered

The proposed requirements template provides a systematic approach for eliciting and documenting requirements, which improves the user's confidence that all special cases have been considered, where special cases are defined as those exceptional cases that differ from the normally expected behavior. In particular, the use of tabular expressions aids in detecting special cases, as shown by Table 2. This table shows how the exceptions of divide by zero errors can be avoided by carefully documenting what input values cause problems for which cases. Tables have the

Table 2. Excerpt from Table for Solution of Unknown External Forces.

| | | $H_1$ | |
|---|---|---|---|
| | | $S_{\text{GET}} = S_{sym} - S_{unkF}$ | $S_{\text{GET}} \neq (S_{sym} - S_{unkF})$ |
| $S_{unkF} \notin \mathbb{P}_3$ | – | $(ErrorMsg' = InvalidUnknown) \wedge ChangeOnly(ErrorMsg)$ | |
| $S_{unkF} = \{@F_{ax}, @F_{bx}, @F_{ay}\}$ | – | $ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$ | |
| $S_{unkF} = \{@F_{ax}, @F_{bx}, @F_{by}\}$ | – | $ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$ | |
| $S_{unkF} = \{@F_{ax}, @F_{bx}, @F_1\}$ | – | $ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$ | |
| $S_{unkF} = \{@F_{ax}, @F_{bx}, @F_2\}$ | – | $ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$ | FALSE |
| $S_{unkF} = \{@F_{ax}, @F_{ay}, @F_1\}$ | $x_1 \neq 0 \wedge \theta_3 \neq 0 \wedge \theta_3 \neq 180$ | $F'_{ax} = \frac{-\cos\theta_3 F_2 x_2 \sin\theta_4 + \cos\theta_3 F_{by} L + F_2 \cos\theta_4 x_1 \sin\theta_3 + F_{bx} x_1 \sin\theta_3}{x_1 \sin\theta_3}$<br>$\wedge F'_{ay} = -\frac{F_2 x_2 \sin\theta_4 - F_{by} L - F_2 \sin\theta_4 x_1 + F_{by} x_1}{x_1}$<br>$\wedge F'_1 = \frac{-F_2 x_2 \sin\theta_4 + F_{by} L}{x_1 \sin\theta_3} \wedge ChangeOnly(S_{unkF})$ | |
| | otherwise | $(ErrorMsg' = Indeterminate) \wedge ChangeOnly(ErrorMsg)$ | |
| $S_{unkF} = \{@F_{ax}, @F_{ay}, @F_{by}\}$ | $L \neq 0$ | $F'_{ax} = F_1 \cos\theta_3 + F_2 \cos\theta_4 + F_{bx}$<br>$\wedge F'_{ay} = \frac{-(-F_1 \sin\theta_3 L - F_2 \sin\theta_4 L + F_1 x_1 \sin\theta_3 + F_2 x_2 \sin\theta_4)}{L}$<br>$\wedge F'_{by} = \frac{F_1 x_1 \sin\theta_3 + F_2 x_2 \sin\theta_4}{L} \wedge ChangeOnly(S_{unkF})$ | |
| | otherwise | $(ErrorMsg' = Indeterminate) \wedge ChangeOnly(ErrorMsg)$ | |
| … | … | … | |
| $H_2$ | | $G$ | |

advantageous property that they can be mathematically verified to ensure that the space that they cover is complete.

Table 2 is an extract from the full table, given in [21], that shows BAS's required behaviour for solving for unknown forces, given the correct number of known forces. Only the statically determinant beam is considered by BAS. Thus the number of unknowns should be the same as the number of equilibrium equations; that is, there should be exactly three unknowns. The valid unknown force space can be defined by $\mathbb{P}_3$, where $\mathbb{P}_3 \triangleq \{S \mid S \subseteq S_F \wedge \sharp S = 3\}$, with $S_F$ being a set of symbols of force variables: $\{@F_{ax}, @F_{ay}, @F_{bx}, @F_{by}, @F_1, @F_2\}$. The @ symbol is used to distinguish between the symbol for a variable and the actual value of the variable. The $\sharp$ symbol is a unary operator that is applied to a set and returns the set's cardinality. The user of BAS should specify three known forces from the set $S_F$, and BAS will solve for the remaining three. The solutions for each element in the set $\mathbb{P}_3$ ($C_6^3 = 20$ situations) are specified in [21]. (An advantage for this problem is that the closed-form solution is available to be analyzed before design.) Some of

the cases may lead to an infinite number of solutions, which come from statically indeterminate beams. Since BAS is only interested in the cases that have a unique solution, the indeterminate situations should be avoided. Table 2, which shows a portion of the full table, documents how the forces are solved for some of the cases. Table 2 uses $S_{unkF}$ to represent a set of symbols of the unknown force variables: $S_{unkF} \in \{S \mid S \subseteq S_F\}$. In Table 2, $S_{sym}$ is the set of symbols of all user interface variables that are either inputs or outputs of BAS. The table specifies that all symbols, except those in $S_{unkF}$, must be in set $S_{\text{GET}}$ before the calculations proceeds. As Table 2 shows, divide by zero errors can be avoided in the implementation by checking (in exact arithmetic) the predicates in the second column of $H_2$.

### 4.4. Catalyzes Consideration of Issues Before Design

The requirements template not only encourages a systematic process for collecting, analyzing and documenting the requirements, it also improves the systematic application of the scientific method shown in Figure 2. There are sections in the requirements template that encourage the analysts to answer questions that will pay significant dividends during the design stage. For instance, the requirements template has a section (SRS Section 4.c.ii) where the sensitivity of the model is considered. The SRS template encourages the analyst to consider the sensitivity of the mathematical model in advance of coding. If the mathematical model is very sensitive to input data errors, then it may not be worth constructing, or it may be necessary to approach the problem in a different manner. For instance, if the beam considered in this case study experiences a high axial load, then buckling may occur. In the current case study an assumption is made that buckling failure will not occur, but if this is an important issue, according to the range of parameters permitted by the model, then a buckling analysis may be appropriate.

Besides the section on sensitivity, there are other sections of the SRS template (Figure 4) that encourage the analyst to think about important issues in advance of the design. These sections are all in the "Non-functional Requirements" section of the template. The achievement of non-functional requirements often involve trade-offs between system qualities, such as accuracy, maintainability, and efficiency in speed and storage, so it is valuable to start the discussion on these trade-offs early in the process. A requirements document allows an estimate of developmental and operational costs, which supports decision making and risk-return analysis. If adequate effort is placed into documenting what behaviour is expected of the system, then it should be easier to judge if the results produced by the system are reasonable. One particular requirement that the analyst should think about in advance of solving the problem is the tolerance allowed in the solution (Section 4.c.iii in the proposed SRS template). For the case study SRS the tolerance is specified by equations of the form $|\sum F_{xi}| \,/\, \sqrt{\sum F_{xi}^2} \leq \varepsilon$, where $F_{xi}$ is the $i$th force component in the $\mathbf{x}$ direction, and $\varepsilon$ is the allowed tolerance. Another valuable set of pre-design non-functional requirements involves the SRS section "Solution Validation Strategies."

Some possible validation strategies for numerical solutions, which are given in the case study SRS for BAS, include: solving the problem by different techniques (such as electronic spreadsheet or a graphical solution) and substituting the calculated results back into the original governing equations to calculate the residual error.

A good design should take into account the possible changes that the system may undergo in the future. This is accommodated in the proposed template by Section 6 of the SRS, "List of Possible Changes in the Requirements." Some potential changes to the case study SRS include incorporating more than two applied forces, considering beams with other types of supports and considering the self-weight of the beam. The designer of BAS will be able to use this information to produce a system that can potentially have a long life because it will be able to evolve to accommodate the likely future changes.

### 4.5. Reduces Ambiguity

One significant benefit of an SRS for an engineering computation problem, such as the beam analysis problem, is that it can reduce the ambiguity of the requirements. By explicitly documenting the requirements and by formalizing some of them, it becomes much easier for different experts to communicate, a review of the requirements is possible and the eventual designer will not have to make arbitrary judgments about the system's required behaviour. One approach to making the SRS unambiguous is to use tabular expressions. As Table 2 shows, tabular expressions clearly specify the required behaviour for all cases when they are complete, which is a property that can be automatically verified. The SRS also has sections that are devoted to providing the necessary details to make the problem unambiguous. For instance, important reference material is given in the SRS sections "Table of Symbols" and "Abbreviations and Acronyms."

Two other sections that reduce ambiguity are "Scope of the Software Product" and "System Context." These sections are important because they respectively delineate what the system does and how the system fits into its external environment. For BAS, the scope is solving for unknown forces, shear, bending moment and deflection of a beam, but the documentation specifies that BAS does not solve for internal axial forces. The scope also makes it clear that BAS will be used for educational purposes. The system context section, on the other hand, documents the user responsibilities, such as preparing input information and using consistent units, versus the system responsibilities, such as detecting data type mismatch and determining if the inputs satisfy the required constraints.

When developing a new system, it is important to have the characteristics of the future users in mind. A beam analysis program probably will be designed differently for a practicing engineer versus a high school student. In the current case study, the decision was made, as documented in the SRS section "User Characteristics," to assume that the users of BAS are first or second year university students in science or engineering.

An SRS is not meant to be read sequentially. Instead it is a reference document that will be searched for specific pieces of information. For instance, a reader or reviewer may search for the definition of a particular term. Without the definition of the term, the reader will not know how to interpret the documentation. For this reason there are two sections in the SRS devoted to definitions: "Terminology Definition" and "Data Definitions." The terms defined include applied force, bending moment, deflection, equilibrium, free-body diagram, longitudinal centroid plane, magnitude of a vector and Young's modulus. The data definitions are used to define the mathematical variables that model objects in the physical system. For instance, there are definitions for the coordinate system, the dimension system, the beam, the reactive forces, the moments and the shear forces. These definitions remove ambiguity by giving a meaning to the symbols and by defining potentially confusing details, such as the sign conventions.

The advantage of unambiguous requirements is that they can end arguments about different designs. Practitioners argue over the relative merits of different designs based on their own implicit requirements. A developer may criticize a design because it is not efficient, but this criticism would not be justified if the designer clearly started out with requirements that clearly stated that precision, maintainability and portability are more important than efficiency. The specific trade-offs depend on the scope of the system. For instance, in the beam analysis case study the requirements for speed and accuracy do not need to be that strict because BAS is not a real-time safety critical application, but rather an educational one.

### 4.6. Range of Model Applicability is Identified

A significant benefit of appropriate and rigorous requirements documentation is that the range of the physical model's applicability can be clearly specified. One way that this is done is by documenting the assumptions that the model is based on, as discussed in the next section. Another way that the range of applicability is identified is by explicitly constraining the input data. The input data should be constrained so that it is physically meaningful. In some cases the software may accept inputs outside of this range, but the user should be warned that the outputs may be incorrect since the assumptions and the theory of the physical model are no longer necessarily valid. In the SRS section "Data Constraints," physical constraints are applied to all variables, as appropriate. For instance, in BAS the location $x_1$ of the applied load $F_1$ is required to be in the range $0 \leq x_1 \leq L$ because any other value is physically meaningless. Another constraint, which is added to maintain the physical system description that the beam is long and thin (PS1.a), is that $0 < h \leq 0.1L$.

Other constraints are added to the input data as system constraints, which are constraints that are not motivated by physics, but rather by the range of values it is reasonable to expect in practice. Table 1 gives an example of system constraints

on the variable $x_1$. BAS also constrains the angle $\theta_3$ so that $0 \leq \theta_3 \leq 180$. This is not necessary for the physics of the problem, but allowing this range of angles, and a signed magnitude for the force, allows one to enter any possible loading, so the system restriction simplifies the user input, without putting any constraints on the range of problems BAS can solve. Another example of a system constraint is that BAS specifies minimum values for some of the inputs, such as $h > h_{\min}$, where $h_{\min}$ is a parameter for the minimum beam thickness. In [21] the specification of BAS's behaviour for each input variable is given, along with a new tabular composition operation that allows the specifications to be combined, while still maintaining the property of domain coverage.

Restricting the input to the system to reasonable values can potentially avoid error cases, and it can streamline the subsequent design stage. One of the hardest challenges in engineering computation is to devise algorithms for a general case. Why should a designer have to face this challenge, if information is known about the problem that will mean that only specific cases will ever occur? For instance, numerical problems could occur in the beam problem if the external forces have significantly different magnitudes, but this is not likely to occur in an engineering problem because the effect of the smallest magnitude forces would be negligible and typically not even modeled by an engineer. For this reason, the case study SRS has constraints on all of the forces of a form like the following constraint on $F_{ax}$:

$$(\min_f \leq |F_{ax}| \leq \max_f) \wedge$$
$$\left[(|F_{ax}| \neq 0) \Rightarrow \forall \left( FF \mid @FF \in S_F \cdot FF \neq 0 \wedge \frac{\max\{|F_{ax}|, |FF|\}}{\min\{|F_{ax}|, |FF|\}} \leq 10^{r_f} \right) \right],$$

where $\min_f$ and $\max_f$ are the system constraints for the minimum and maximum magnitude forces, and $r_f$ is a positive integer that is the maximum exponent of base 10 for the ratio between the magnitudes of the largest and smallest forces.

Documenting the range of applicability of the model is also important for engineering computation problems other than the beam analysis problem. For instance, it is difficult to write code to solve any system of equations $\mathbf{Ax} = \mathbf{b}$, but this job becomes easier if $\mathbf{A}$ is known to have special characteristics, such as being symmetric positive definite. Therefore, the requirements documentation should clearly show the restrictions on the data and the theory that will lead to the simplifications.

### 4.7. Clear Documentation of Assumptions

In engineering computation it is often the differing assumptions that distinguish one piece of work from another. Assumptions are necessary to build a physical model of the real world. Often the quality of the model depends on how reasonable the simplifying assumptions are. Given the importance of assumptions, an SRS for engineering computation problems should clearly label and document them, as in Section 4.b.i of the proposed template. The assumptions defined in this subsection

simplify the original problem and fill in missing information for the physical system so that a theoretical models can be developed and be properly applied. Some sample assumptions for the beam analysis SRS, using the numbering given in [21], include the following:

**A1.** The physics for this problem are in the field of Newton's classical mechanics.

**A2.** Thermal effects are neglected.

**A4.** The weight of the beam is neglected.

**A8.** Only the statically determinant cases for the beam are considered.

**A9.** Beam deformations are small compared to the original dimensions. Thus the slope of the beam's deflected shape is small compared to unity (slope $\ll 1$), and the length variation of the beam in the longitudinal direction is neglected.

**A10.** The deflection of the beam is caused by bending moment only; the shear does not contribute.

**A14.** The second moment of area along the length of the beam is constant.

Since different models are distinguished by their differing assumptions, a clear indication of the dependence of the model on its assumptions will greatly facilitate reuse and future evolution of a given SRS. When the assumptions change the model generally changes, but very often some parts will stay the same. By identifying those parts one can easily reuse them. In the proposed SRS template the interdependence of the different components of the specification is documented by a traceability matrix, which is in the SRS Section 5. For the traceability matrix to be meaningful it is important that the assumptions be independent of one another.

The traceability matrix gives a "big picture" view of the relationships between sections "Physical System Descriptions," "Goal Statements," "Data Definitions," "Assumptions," "Theoretical Models" and "Instanced Models" in the case study SRS. These sections of the SRS's are shown because they contain the most essential information of the system functionalities. As an example, the relationship between "Goal Statements" and "Theoretical Models" can be refined by a specific relationship between one concrete goal statement and one concrete theoretical model, for instance **G1** and **T1**. These relationships are represented by ticks "$\surd$" in the cells of the matrix, as in Table 3, which is a portion of the table presented in [21]. The physical system descriptions and the goals in the first column of the matrix are used by the data definitions or the theoretical models in the second column; the assumptions in the first row are used by the theoretical models, the instanced models and the data definitions in the second column; the theoretical models and the data definitions in the second column are used by the instanced models in the first row.

Table 3. Traceability Matrix for Deformed Beam.

| Phy. Sys. | Data | Assumption | | | | | | | | | | Model | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| / Goal | / Model | A1 | A2 | ... | A4 | ... | A8 | A9 | A10 | ... | A14 | **M1** | ... |
| **G1** | **T1** | √ | | ... | | ... | √ | √ | | ... | | √ | ... |
| **G2** | **T2** | √ | | ... | | ... | √ | √ | | ... | | | ... |
| **G3** | **T3** | √ | | ... | | ... | | √ | √ | ... | | | ... |
| | **M1** | | √ | ... | | ... | | | | ... | | √ | ... |
| PS1.a | *L* | | | ... | | ... | | | √ | ... | | √ | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

The traceability matrix of the "deflected beam problem" presented in this paper can be compared with the simpler case of a "rigid beam problem" to illustrate how requirement reuse can be implemented with the help of a traceability matrix. Compared with the deflected beam, the rigid beam has a new assumption:

**A15.** The beam behaves as a rigid body.

This new assumption will require several changes to the SRS for the deflected beam. The modifications can be guided by the traceability matrix. To see this, one can consider how the flexible beam traceability matrix can be modified to obtain a traceability matrix for the rigid beam problem. The addition of the rigid body assumption means that the previous assumptions **A9**–**A14** are no longer necessary. The new system does not require system goals **G2** and **G3**; therefore, theoretical models **T2** and **T3** will be removed from the row headers of Table 3. Removing the columns representing the unnecessary assumptions and the rows of the unnecessary models and symbols results in a traceability matrix that reflects the requirements specification of the "rigid beam problem" exactly. The details of this transition can be found in [21].

Besides supporting a transition to a simpler model, the traceability matrix also supports a move to a more complex model. For instance, assumption **A10** could be removed, which would mean that shear would also contribute to deflections and a more complex theory, such as Timoshenko beam theory [34, pp. 224–230] would be required. In the transition to the more complex system the traceability matrix would guide the analyst in determining what sections of the documentation need to be added and/or modified. For instance, Table 3 shows that the theoretical model **T3**, which is the model for simulating the deflection, will need to be changed. Also, PS1.a, which says that the "beam is long and thin" is no longer necessarily true as Timoshenko beam theory also applies to short and thick beams.

## 5. Concluding Remarks

This paper motivates, justifies and illustrates a method of writing requirements specifications for engineering computation problems that will improve their reliability. The motivation comes first from the fact that reliability of a system can

only be accurately judged if there is an unambiguous statement of the behaviors and qualities that the system is required to have. Although there are many excellent numerical libraries and packages that implement accurate and efficient algorithms, the selection of an appropriate algorithm is often based on implicit requirements. If explicit, appropriate and rigorous requirements documentation is available, then the overall quality of the design can be improved, not just with respect to reliability, but also with respect to usability, verifiability, maintainability, reusability and portability, which are sometimes neglected qualities in engineering software. Further motivation for the appropriateness of software engineering methodologies, such as requirements analysis, for engineering computation, comes from the fact that the waterfall model of software development closely parallels the usual model of the scientific method.

The justification for using the proposed template for engineering computation comes from observing how it supports and encourages a systematic process by providing guidelines, by providing a smooth transition from general to specific details, by increasing confidence that all special cases have been considered, and by encouraging the analyst to scrutinize their problem in advance of designing the computational system. Further justification for using an SRS for engineering computation comes from the benefit of reducing ambiguity, clearly identifying and documenting the range of model applicability and rigorously documenting the assumptions that simplify the real world to the point where theoretical and instanced models can be constructed. To improve the systematic process and to reduce ambiguity, this paper advocates the use of tabular expressions, which provide mathematical rigor, but at the same time have the benefit that they can be easily and intuitively understood.

To illustrate the proposed method of documenting requirements for engineering computation problems, this paper presents a case study for the analysis of a statically determinant beam. Although this problem is relatively simple, the findings of this study can be generalized because many engineering computation problems follow the same pattern. For instance, the scientific method is appropriate for most engineering problems and engineering software can often be abstracted by the following simple sequential model: accept information, perform calculations, and report results. Moreover, the calculation step is similar between many engineering computation problems because it involves solving some given set of governing equations together with appropriate boundary and/or initial conditions. Given the similar pattern between the beam problem and other engineering problems, the method presented in this paper can be applied to larger and more complex problems. The advantages of the current method will greatly increase as the size and complexity of the problem grows because the value of a systematic approach increases with the number of details and the number of people involved. To demonstrate the advantages of the current method in the context of engineering computation, future studies should be conducted to obtain empirical data comparing the new approach with traditional methods.

## Acknowledgements

The financial support of the Natural Sciences and Engineering Research Council (NSERC) of Canada is gratefully acknowledged. The authors would also like to thank the anonymous referees for their thoughtful and valuable comments.

## References

1. Åhlander, K., Haveraaen, M., and Kaas, H. Z.: On the Role of Mathematical Abstractions for Scientific Computing, in: *The Architecture of Scientific Software*, Kluwer Academic Publishers, Boston, 2001, pp. 145–158.
2. Berry, D. M.: Formal Methods: The Very Idea, *Science of Computer Programming* **42** (2002), pp. 11–27.
3. Blilie, C.: Patterns in Scientific Software: An Introduction, *Computing in Science and Engineering* **4** (3) (2002), pp. 48–53.
4. Davis, A. et al.: Identifying and Measuring Quality in a Software Requirements Specification, in: *Proceedings of the 1st International Software Metrics Symposium*, IEEE, 1993, pp. 141–152.
5. Davis, A.: *Software Requirements: Objects, Functions, and States*, Prentice Hall, Upper Saddle River, 1993.
6. Desharnais, J., Khedri, R., and Mili, A.: Representation, Validation, and Integration of Scenarios Using Tabular Expressions, *Formal Methods in System Design*, accepted for publication.
7. DOD: *Software Development and Documentation, DOD Military Standard MIL-STD-498*, Technical Report, US Department of Defence, Washington, 1994.
8. Dubois, P. F.: Designing Scientific Components, *Computing in Science and Engineering* **4** (5) (2002), pp. 84–90.
9. ESA: *ESA Software Engineering Standards, PSS-05-0 Issue 2*, Technical Report, European Space Agency, 1991.
10. Galin, D.: *Software Quality Assurance: From Theory to Implementation*, Pearson Education Limited, 2004.
11. Gerlach, J.: *Domain Engineering and Generic Programming for Parallel Scientific Computing*, PhD thesis, Von der Fakultät IV—Elektrotechnik und Informatik der Technischen Universität Berlin, 2002.
12. Ghezzi, C., Jazayeri, M., and Mandrioli, D.: *Fundamentals of Software Engineering*, 2nd edition, Prentice Hall, Upper Saddle River, 2003.
13. Grundy, J.: Aspect-Oriented Requirements Engineering for Component-Based Software Systems, in: *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on Limerick, Ireland, June 7–11 1999*, Institute of Electrical and Electronics Engineers, pp. 84–91.
14. Hannemann, R., J.H., Zellerhoff, M., and Klinkenbusch, L.: Scientific Programming in Field Theory, Part 1, *IEEE Computing in Science and Engineering* **3** (3) (2001).
15. Heninger, K. L.: Specifying Software Requirement for Complex System: New Techniques and Their Application, *IEEE Transactions on Software Engineering* **6** (1) (1980), pp. 2–13.
16. Heninger, K. L., Kallander, J., Parnas, D. L., and Shore, J. E.: *Software Requirements for the A-7E Aircraft*, NRL memorandum report, United States Naval Research Laboratory, Washington, 1978.
17. IEEE: *Recommended Practice for Software Requirements Specifications, IEEE Std. 830*, IEEE, 1998.
18. Janicki, R. and Khedri, R.: On a Formal Semantics of Tabular Expression, *Science of Computer Programming* **39** (2–3) (2001), pp. 189–213.
19. Kaindl, H., Brinkkemper, S., Bubenko, J. A., Jr., Farbey, B., and Greenspan, S. J.: Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda, *Requirements Engineering Journal* **7** (3) (2002).
20. Kreyman, K. and Parnas, D. L.: *On Documenting the Requirements for Computer Programs Based on Models of Physical Phenomena*, SQRL Report 1, Software Quality Research Laboratory, McMaster University, 2002.

21. Lai, L.: *Requirements Documentation for Engineering Mechanics Software: Guidelines, Template, and a Case Study*, Master's thesis, McMaster University, Hamilton, 2004.
22. Mylopoulos, J., Chung, L., and Yu, E.: From Object-Oriented to Goal-Oriented Requirements Analysis, *Communications of the ACM Archive* **42** (1) (1999), pp. 31–37.
23. NASA: *External Interface Requirements DID, SMAP-DID-P210, release 4.3*, Technical Report, National Aeronautics and Space Agency, 1989.
24. NASA: *Software Requirements DID, SMAP-DID-P200-SW, release 4.3*, Technical Report, National Aeronautics and Space Agency, 1989.
25. Parnas, D. L.: *Tabular Representation of Relations*, CRL Report 260, Telecommunications Research Institute of Ontario (TRIO), McMaster University, 1992.
26. Parnas, D. L. and Clements, P. C.: A Rational Design Process: How and Why to Fake It, *IEEE Transactions on Software Engineering* **12** (2) (1986), pp. 251–257.
27. Parnas, D. L., Asmis, G. J. K., and Madey, J.: Assessment of Safety-Critical Software in Nuclear Power Plants, *Nuclear Safety* **32** (2) (1991), pp. 189–198.
28. Roache, P. J.: *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque, New Mexico, 1998.
29. Robertson, S. and Robertson, J.: *Mastering the Requirements Process*, ACM Press/Addison-Wesley Publishing, New York, 1999, chapter "Volere Requirements Specification Template", pp. 353–391.
30. Sanga, B.: *Assessing and Improving the Quality of Software Requirements Specification Documents (SRSDs)*, Thesis for M.Sc. program, Computing and Software Department, McMaster University, Hamilton, 2003.
31. *Scientific Discovery through Advanced Computing*, Office of Science, U.S. Department of Energy, 2000, *http://www.osti.gov/scidac/*.
32. Sommerville, I. and Sawyer, P.: *Requirement Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.
33. Thayer, R. H. and Dorfman, M. (eds): *IEEE Recommended Practice for Software Requirements Specifications*, 2nd edition, IEEE Computer Society, Washington, 2000.
34. Timoshenko, S. and Young, D. H.: *Elements of Strength of Materials*, 5th edition, D. Van Nostrand Company, 1968.
35. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour, in: *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, Washington, 2001, pp. 249–263.