On Documenting the Requirements for Computer Programs Based on Models of Physical Phenomena

Konstantin Kreyman and David Lorge Parnas

Abstract

Programs for use by Scientists and Engineers are usually embodiments of mathematical models of physical phenomena. Complete and accurate models are usually quite complex because they must deal with the wide-variety of situations that can arise in the real-world. Informal descriptions of these models are often incomplete, imprecise, and, inaccurate and are not suitable for specifying what is required of a software package. This paper presents an approach to writing requirements documents for such programs. It demonstrates how tabular notation can make precise mathematical expressions more readable. It also shows how we can document systems in which the user is given some control of the computational method to be used.

1 Why do we need a better design process for software that models physical phenomena?

For many decades computers have been used to predict the behaviour of physical systems by using software that implements mathematical models of physical processes. These predictions can be very important as they are often used in the design of critical structures and systems, evaluating environmental impact, or controlling potentially dangerous processes. Unfortunately, often the predictions are wrong because the software is not an accurate implementation of the model of the physical phenomena. Errors in the software are often quite hard to detect because the underlying model may not be accurately and completely described; the software is quite complex and differences between the intended model and the actual one may be quite subtle [11]. This paper is part of an ongoing effort to develop methods and notations that improve the quality of software and reduce the number of errors made in its development.

The most effective way to avoid faults during a system's operation is to eliminate or reduce errors during the initial design and early development phases. It is much more difficult to correct those errors later, when the design has progressed and the complexity has increased.

Although there have been some successes in solving complex engineering and scientific problems using computer systems, the usual process of designing software appears quite irrational to many observers. Programmers usually start working without a clear statement of the desired behaviour and implementation constraints and often solve the wrong problem.

Ideally, we would derive computer programs from a statement of requirements in the same way that theorems are derived from axioms in mathematical proofs. Probably, we'll never see a software project that proceeds in such a perfectly "rational" way. Parnas and Clements [16] discussed at least seven reasons why that would be hard to achieve. However, precisely documenting the requirements can improve the software design process in many applications. Without a proper requirements document, we may end up with a good solution to the wrong problem. Program developers are also quite likely to overlook important special cases or even to invest effort in modelling situations that are not important.

2 Why do we need a relational model of documentation?

A significant portion of most software development efforts is devoted to documentation [1, 14]. In addition to improving program quality, good documentation improves software usability and maintainabiltiy; these are very important for reducing long-term software related expenses.

Although there is widespread recognition that documentation is important, the literature often contains different, even contradictory, opinions about whether or not a certain fact should be included in a given software document. Most documentation standards specify the format, but not the information content of the document. For these reasons, similar information is often included in several documents or not found in any.

Documentation can play a major positive role in the software development process. Unfortunately, very often, the documentation is one of the final steps in software development [19]. If the documentation is written by the developers when the project is nearing completion, the document is often useful to people who know the software well, but usually hard for others to understand. Because of the complexity of mathematical models that are usually used in engineering and scientific simulations, any way to make requirements documentation easier to understand and use during software development could dramatically reduce the number of errors found in such software [11].

A positive industrial experience with relational documentation is reported on the results of Bell Labs project [7]. The time invested in development of a precise statement of requirements was more than regained when the system received its first on-site test. The test period was the shortest in the Lab's history. Software assessment showed that most of the misunderstandings that normally become apparent during testing had been eliminated when the relational requirements document was reviewed. Another result of the rational approach was improved accuracy and usability of informal user documentation. It was found that if the structure of the informal documentation was based on the structure of formal software documentation and, in fact imitates it, the quality of the informal documents was greatly enhanced. The use of formal mathematical documentation to supplement the informal user documentation is consistent with accepted engineering standards.

A relational requirements model and program documentation model were used for inspecting of safety-critical computer programs for the Darlington Nuclear Power Generating Station in Ontario (Canada). At Darlington, the safety shutdown systems were computer controlled. The information that was actually relevant to understanding the shutdown system of nuclear reactor was summarized in a system requirements document. They described exactly what properties are needed for the shutdown system [15]. The relational specifications of the Darlington Shutdown Software allowed a team of engineers and programmers to work together when inspecting the code. The precise specification was essential for understanding, effective inspection and testing and correcting the software.

This paper outlines an approach to software documentation and software design that is intended to help scientists, engineers and programmers who develop or review software to deal with the complexity of software products. If software that is based on mathematical models and destined for predicting physical phenomena or behaviour of a system is to play a safety-critical role, it is essential to restrict the complexity of the programs. We believe that the technique described below can not only simplify the process of software requirements development but will also help all members of the team to obtain a more complete understanding of the phenomena being modelled. It will combine the knowledge of model and program developers and thereby contribute to the improvement of software quality in many engineering and scientific applications.

3 Mathematics-based models of requirements documents: basic definitions

Although academic computer science has investigated formal methods intensively, such methods are not routinely used in industry. Even when formal methods are often used, they are

used only to a limited extent and resisted by engineers and programmers. This situation is hardly surprising, both because there are many widely held misconceptions about the use of formal techniques [2] and because many of the methods are not practical. Below we will argue that it is possible, to obtain benefits from methods based on a relational model of software requirements.

The principles of documentation and the software design process most applicable for our objectives are based on the concept that the core of a requirements document is a mathematical model, including a set of mathematical relations that may be described in tabular form [6, 16, 20]. Assuming that each document must contain a representation of one or more binary relations, and that each relation describes a set of ordered pairs[a], Parnas and Madey [17], defined the contents of a number of standard documents, including requirements documents. One of these documents is considered to be complete if it contains enough information to determine whether or not any ordered pair should be included in the relation. This approach automatically answers questions frequently asked by programmers and engineers about what information should be included in a document. In this approach, a document containing additional information is considered to be faulty, because misinterpretation of the additional information is possible [5].

The four-variable model works well for control systems [15] and has also been used for information systems [7] but does not fit well for programs that are based on mathematical models of specific phenomena. This paper presents a five-variable model that better reflects the reality of forecasting tools and makes it easier to apply relational methods in practice.

3.1 The environmental variables

The choice of quantities describing the system under consideration is a critical step in the development of any mathematical model. As in [17], the requirements analysis begins with identification of the environmental quantities. The environmental quantities are divided into two sets: *monitored quantities,* MQ, i.e. those that the user wants the computer system to observe and measure, and *controlled quantities,* CQ, whose values the software is intended to compute (predict). Monitored quantities are physical quantities that exist outside of the computer system. Controlled quantities are also physical quantities that are measurable outside of the computer system but may be incorrectly perceived as being part of the system because the system controls them[b]. Although these are concepts that are often used in the design of real-time control systems, they are also useful concepts for designing other systems such as Computer Aided Design (CAD) systems [21].

The monitored quantities can be further partitioned into those whose values are unknown and should be computed and those whose values are known or can be measured. Often, the controlled quantities are intended to be accurate estimates of the monitored quantities that cannot be measured directly. Below we will use $m_1,m_2,..m_p$ and $c_1,c_2,,..c_q$ to denote quantities from sets MQ and CQ, respectively. The values of each of these quantities can be recorded as functions of time. The function describing the value of an environmental quantity a at time t will be denoted by a^t. The tuple of time-functions $(m^t_1,m^t_2,..m^t_p)$ and $(c^t_1,c^t_2,\ldots c^t_q)$, containing one element for each of the monitored or controlled quantities will be denoted by m^t and c^t respectively.

[a] For example the relation y = x + 1 is a set of pairs of integers (x,y) that includes... (1,2) (2,3)... (99,100)...

[b] For example the clock on a computer screen is a physical quantity measurable outside the system.

3.2 The computer's inputs and outputs

Input devices are used to sense the values of the monitored variables and transform them into digital inputs for the computers. The set of inputs will be denoted $i_1,i_2,..i_w$. Similarly, output devices transform digital output values to values of the controlled variables. The set of outputs will be denoted $o_1,o_2,..o_r$. The inputs provide information about the values of the monitored quantities while the outputs influence the values of the controlled quantities.

3.3 Method dependant values and user preferences

In many scientific and engineering applications, numerical methods are applied to obtain a solution of a system of equations that constitutes a mathematical model of the physical phenomena under consideration. It is important to distinguish between a model and a method. A mathematical *model* describes the physical phenomena in terms of the values of physical variables in that environment. A model may embody any of the physical laws including special properties of materials and the environment. To compute the values described by the model, we must choose a *computational method*, i.e a specific procedure for calculating values that are (approximately) those predicted by the model. For any model, there will be several methods that we might consider.

When we apply a computational method, we usually introduce new quantities that do not correspond to any quantities in the environment and are not mentioned in the model. These quantities are method dependant, that is, if we choose a new method, we may have different quantities. Examples of such method-dependent quantities are mesh size, termination conditions for iterative procedures, the resolution of numeric variables, etc. These method-specific quantities are not properties of the physical world; they only have meaning in reference to a specific computational method.

We may also introduce quantities that represent the user's assumptions about the environment and their preferences about the results. Examples of characteristics that a user may be allowed to specify are accuracy and output report format. Nothing in the physical problem deals with these values. For example, mathematical models do not deal with inaccuracy caused by finite numerical representations - all values in the model are the ideal values. However, limitations on our computational devices and methods will introduce error and a user may wish to specify upper bounds on that error. Examples of user assumptions about the environment include limitations on the range of values or their rate of change.

In an earlier approach known as the four-variable-model, [15, 17, 21], such model-dependant variables were treated as if they were environmental variables but this can be confusing. It confuses the distinction between model and method, and can lead designers to ignore important factors and to make poor software design decisions. Distinguishing these non-environmental quantities from the environmental monitored quantities has many advantages. For example, it would allow a developer to use new methods without having to make changes in most of the documents describing the system requirements.

The values of these non-environmental quantities must be communicated to the software. We will refer to the non-environmental quantities that are described above as $n_1, n_2,..n^c_g$. In the four-variable model, the inputs *i*, provide information about the values of *m*. In this paper, we extend

[c] In principle the non-environmental quantities can change while the system is running and should be represented as time functions. However, this is not the usual case and we will not do so.

the set of inputs to include additional items that will carry information about elements of n. The additional inputs to the software, $\underline{i}$, will be a function, NEQ, of (1) method dependant quantities, (2) users assumptions about the values of *m* and (3) user requirement variables, i.e. variables used by the user to state required quantitative properties of the solution. Thus $\underline{i}$ = NEQ(n) where n denotes a tuple specifying values for these non-environmental quantities. Figure 1 shows the computational system and the five variables (m,n,i•$\underline{i}$,o,c) described above.

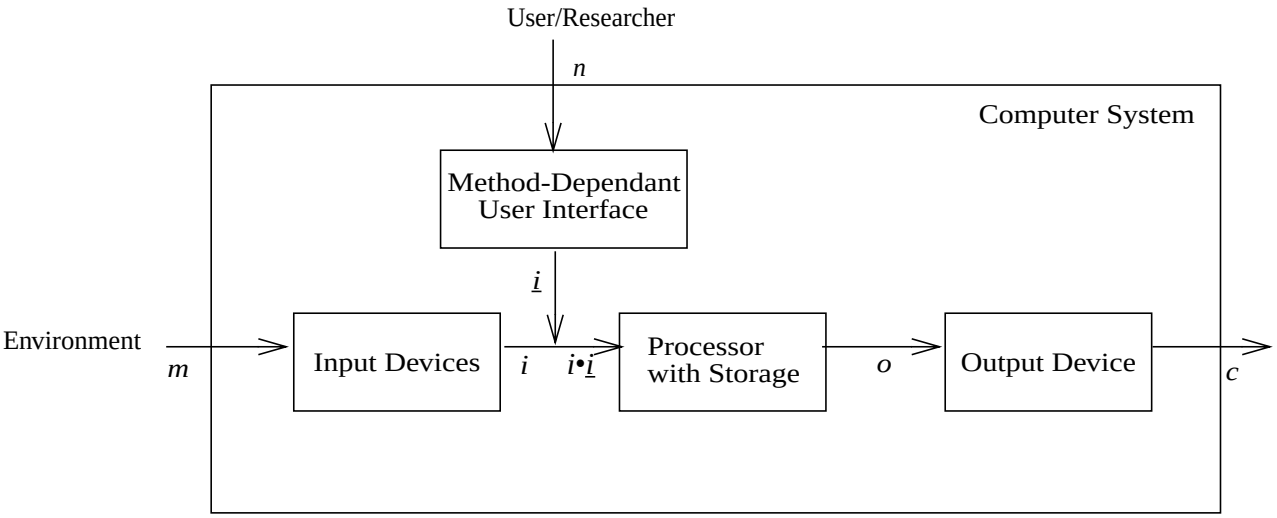

Figure 1: The five variable model of software modelling physical phenomena as explained in 3.

3.4 Relations between the variables

A relation NAT will describe the environmental restrictions, (which result from the constraints imposed by nature and previously installed systems), on the values of environmental quantities. In accordance with the approach taken in [17], NAT will be defined as follows:

- domain (NAT) is a set of vectors of time functions containing exactly the instances of m^t allowed by the environmental constraints
- range (NAT) is a set of vectors of time functions containing exactly the instances of c^t allowed by the environmental constraints

If there are no constraints on m^t and c^t, then NAT = MQ × CQ. However, usually NAT is a subset of MQ × CQ and is not a function. It is important that if any values of m^t are not in the domain of NAT, the program designer may assume that these values will never occur. Thus, a representation of the relation NAT describes the phenomena to be modelled by the software. For example, often NAT could be described by a set of equations. It should be the responsibility of scientists and engineers who are experts in the domain of application of the software to determine the relation NAT. They need not be skilled programmers to do this but they must understand what information the programmer will need to develop a system that will adequately predict the behaviour of the variables of interest.

To restrict the relation NAT to environmental quantities (nature) only, a new relation[d] NEQ, describing newly introduced quantities, can be used.

[d] This relation will usually be a function.

Relation NEQ is defined as follows:

- domain (NEQ) is a set of vectors of time-functions containing the new quantities n^t.
- range (NEQ) is a set of vectors of time-functions containing possible instances of $\underline{i}^t$.
- $(n^t, \underline{i}^t) \in$ NEQ if and only if $\underline{i}^t$ describes values of the input registers that are possible when n^t describes the values of the non-environmental quantities.

Separating NEQ from NAT will make both descriptions clearer. It also clarifies who is responsible for providing the information.

The computer system behaviour that is acceptable can be documented by describing a relation REQ, defined as follows:

- domain (REQ) is a set of pairs of vectors of time functions containing the instances of m^t allowed by the environmental constraints and all possible values of n^t.
- range (REQ) is a set of vectors of time functions containing only those instances of c^t that are allowed by a correctly functioning computer system.

The relation REQ can be considered feasible with respect to the relation NAT and NEQ if

$$\text{domain (REQ)} \supseteq \text{domain (NAT')} \tag{1}$$

$$\text{domain (REQ} \cap \text{NAT')} = (\text{domain(REQ)} \cap \text{domain(NAT')})^{e} \tag{2}$$

$$\text{where NAT'} = \{((m,n),c) \mid (m,c) \in \text{NAT}\}. \tag{3}$$

Feasibility in this sense means that the restrictions described in relation NAT or NAT' and NEQ will allow the required behaviour as described by REQ.

For several reasons, for instance the approximate nature of numerical computations, imperfect computational algorithms or computer hardware, "small" errors are usually acceptable in the results of computations, i.e. in the values of controlled quantities. Therefore the relation REQ is usually not a function. Moreover, the relation REQ may depend on the method dependent quantities.

The interpretation of the inputs can be described by relations IN and NEQ. Before further defining IN, it is convenient to define IN'. Relation IN' is a set of pairs (m^t, i^t). The pair (m^t, i^t) is in IN' if i^t represents possible values of the inputs when m^t represents the values of the monitored qualities. The relation IN can be defined as follows:

- domain (IN) is a set of vectors of time-functions containing the possible instances of $m^t \bullet n^t$ where "•" denotes concatenation;

$$\text{domain (IN)} \supseteq \text{domain (NAT')} \tag{4}$$

- range (IN) is a set of vectors of time functions containing possible instances of $i^t \bullet \underline{i}^t$

Relation IN is a set of pairs of vectors of time functions $((m^t \bullet n^t),(i^t \bullet \underline{i}^t))$. The pair $((m^t \bullet n^t),(i^t \bullet \underline{i}^t))$ is in IN if (m^t, i^t) is in IN' and $(n^t, \underline{i}^t)$ is in NEQ.

The behaviour of the output device can be described by means of relation OUT as follows:

- domain (OUT) is a set of vectors of time-functions containing the possible instances of o^t
- range (OUT) is a set of vectors of time functions containing all possible instances of c^t.

[e] Note that these formulae assume that the values of the newly introduced quantities are independent of the values of m.

The software will provide a system with input–output behaviour. It can be described by a relation SOF. Relation SOF is defined as follows:

- domain (SOF) is a set of vectors of time-functions containing the possible instances of $i^t \bullet \underline{i}^t$
- range (SOF) is a set of vectors of time-functions containing the possible instances of o^t

For the software to be acceptable, SOF must satisfy the following condition[f]:

$$\forall\, m^t,c^t,n^t,i^t,\underline{i}^t,o^t\ [(IN\ (m^t \bullet n^t, i^t \bullet \underline{i}^t) \wedge SOF(i^t \bullet \underline{i}^t, o^t) \wedge OUT(o^t,c^t) \wedge NEQ(n^t,\underline{i}^t) \wedge NAT(m^t,c^t)) \Rightarrow REQ(m^t,c^t)] \quad (5)$$

Figure 2 shows the computing system and the relations described above.

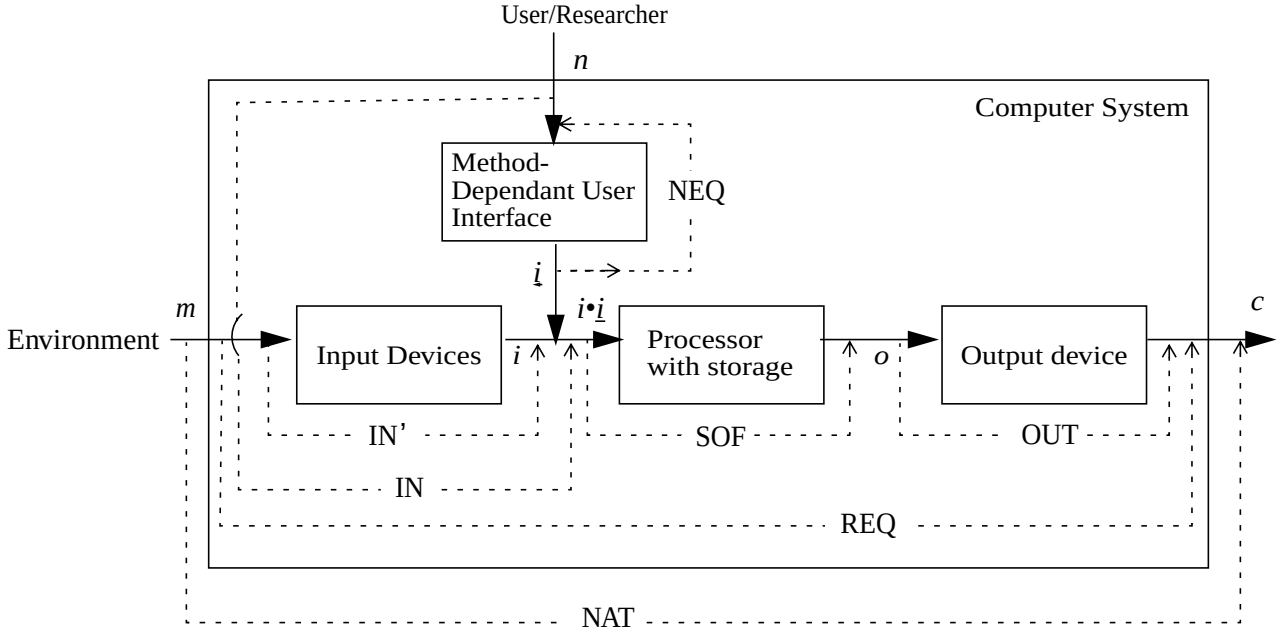

Figure 2: Relational structure of the five variable model. Solid arrows show data flow. Dashed arrows show the relations discussed in 3; *m, n, c, i•i, o* are as in Figure 1.

4 Interaction of team members during the requirements document development process.

The approach in Section 3 allows individual team members to start working on some parts of the documentation independently. Ideally, the relation NAT would be described by professionals who are experts in mathematical modelling of physical phenomena. Computing professionals need not be involved in writing this description, but they must read it.

Development of the relation NEQ requires cooperation between the future users and experts in numerical algorithms and computations. Developing the relation REQ can be done by software system designers consulting with representatives of the future users and should not require an understanding of computational methods. Computer or Software Engineers must describe the relations IN and OUT. The relation IN' combines the work of Software Engineers, future users, and experts in numerical algorithms. The software implementers inspect the relation SOF to determine its implementability, and then use it to guide this work.

[f] From this expression one might identify this model as a six-variable model. We have chosen to consider i and $\underline{i}$ as a single variable.

In theory, this approach allows experts from various disciplines to work independently, but, in fact, they must cooperate to be sure that what they have specified is compatible and feasible. Having an interdisciplinary team produce the documents mentioned above will make it more likely that the software will accurately reflect the physical conditions, and engender more confidence in the accuracy of the computed results.

5 Tabular representation of relational requirements documents: an illustrative example

The use of the relations described above, represented as tables, for documenting software requirements and behaviour has a comparatively short history. In late 70's tabular expressions or tables were used to document aircraft software [6]. The tabular approach was developed in response to the practical needs of industry and government which were experiencing the frustration of trying to get software to do the right thing. Many "bugs" were caused by misunderstandings that would have been avoided by better documenting of software requirements. Tables can be constructed recursively from simpler components, namely conventional expressions and grids [13]. A general syntax and semantics of tables, based on the concept of a cell-connection graph, has been developed more recently [8, 9, 10, 22]. Some promising results of industrial applications of the approach to software that controls systems in "real-time" were obtained in recent years [4]. However, software designed for predicting the behaviour of physical systems has not received the same attention.

When software is designed for predicting physical phenomena, the relation between the monitored quantities can be described by means of a system of equations, which together with necessary initial and boundary conditions becomes the basis of the computer programs. Below, as an example, we consider the radioactive contamination of the ocean. To illustrate our approach, we have simplified the processes and physical interactions in this example as much as possible.

Many pollutants are transferred from the atmosphere to the ocean. Among these pollutants are radioactive elements that appear in the atmosphere because of human activity and can cause radioactive contamination of the ocean. Penetration of radionuclides, deposited on the ocean surface to the deep layers is caused by several factors. Vertical turbulent diffusion is one of the major mechanisms in the long term. We will assume that vertical turbulent exchange is the only mechanism influencing the distribution of a pollutant concentration c in a water body, and the mass of a pollutant isotope of half-life τ, decaying in unit time, is in direct proportion to the total amount of pollutant. Under the assumption that the coefficient of vertical turbulent diffusion k_c does not change along the vertical (z) axis, the general equations together with necessary initial and boundary conditions are those presented in Figure 3. There, t represents time, P_t is a period of time for which computations have to be carried out, q_0 is a contaminant flux through the unit of area of ocean surface per unit of time, c_D is the pollutant concentration at $z=D$, where the water depth D is counted from the free water surface along the z-axis, l is a turbulent path assumed to be constant, $\frac{dT}{dz}$ is the vertical temperature gradient (which is known), k_v and k_T are coefficients of turbulent impulse exchange and turbulent diffusion of heat respectively, μ is a constant coefficient of turbulent exchange reflecting others mechanisms, which differ from shearing turbulence generated by drift currents with horizontal components u and v, g is the acceleration due to gravity, ρ and ρ_a is water and air density correspondingly, α_c, α_T, γ_T, β and c_1 are constants, C_a is a friction coefficient in atmosphere, W is the wind speed, $f = 2\omega \sin\varphi$ is the Coriolis parameter, φ is latitude. In the full document each variable would be listed and defined.

DIFF

t = 0	$P_t > t > 0 \vee t = P_t$	$t < 0 \vee t > P_t \vee$ **¬TURB**

H1

z = 0	c = c_D	$k_c\,(\partial c/\partial z) = q_0 \wedge$ **TURB**	c = -1
0<z<D	c= c_D	$\partial c/\partial t = k_c\,(\partial^2 c/\partial z^2) - \lambda c \wedge \lambda = ln\ 2\ /\tau \wedge$ **TURB**	c = -1
z = D	c= c_D	c = c_D	c = -1
z<0 ∨ z>D	c = -1	c = -1	c = -1

H2 G

TURB

$\psi \le 0 \wedge \psi = (\partial u/\partial z)^2 +$ $(\partial v/\partial z)^2 - \alpha_T \gamma_T g\,(\partial T/\partial z) \wedge$ **CUR**	$\psi > 0 \wedge \psi = (\partial u/\partial z)^2 +$ $(\partial v/\partial z)^2 - \alpha_T \gamma_T g\,(\partial T/\partial z) \wedge$ **CUR**	**¬CUR**

H1

k_T |	$k_T = \mu$	$k_T = (\alpha_T c_1^2 l^2/D) \int_0^D (\psi)^{1/2}\,dz + \mu \wedge$ **CUR**	$k_T = {}'k_T$
k_c |	$k_c = \alpha_c k_T$	$k_c = \alpha_c k_T$	$k_c = {}'k_c$

H2 G

CUR

$z = 0$	$0 < z < D$	z = D	$z < 0 \vee z > D$

H1

u,v |	$k_v(\partial u/\partial z) = -G_0/\rho \wedge$ $k_v\,(\partial v/\partial z) = 0 \wedge$ $G_0 = \rho_a C_a W^2 \wedge$ $k_v = \rho_a C_a\, WH/\beta$	$k_v(\partial^2 u/\partial z^2) + fv = 0 \wedge$ $k_v\,(\partial^2 v/\partial z^2) - fu = 0 \wedge$ $k_v = \rho_a C_a\, WH/\beta$	u = v = 0	u = v = -1000

H2 G

Figure 3: The tabular representation of time-dependent problem for computing pollutant concentrations at the different depths in a water body. An index "'" denotes the value of a variable before calculations. Values -1 and -1000 used as flags to indicate that these are cases where the models do not apply.

The tables presented in Fig. 3 not only provide a computer specialist with information about core equations but also show which of the equations are used for computing pollutant concentrations at a given depth and time. Each table represents a separate process; the composition of these tables embodies a model of the phenomenon under consideration. For

instance, the table DIFF describes the vertical turbulent diffusion of a contaminant in a water body. The table TURB describes a model that we can use to compute the average value of the coefficient of turbulent exchange within a water body. The method for calculating the drift currents in a horizontally boundless ocean is presented in the table CUR. Taken together, the three tables completely describe a model of the phenomenon under study. Even someone who is not a physicist can see the interconnections clearly. The set of interconnected tables constitutes a model of the phenomena in the same way that pieces of coloured glass form a mosaic.

The concise tabular description of the physical processes given above, warns the software developers about cases that are not covered by the model and other possible errors and describes the system behaviour that is required in those cases. For example, the tables in Figure 3, clearly indicate that the logic errors described by [11] can arise in a program when $t<0$, $t>P_t$, $z<0$, $z>D$ because under those conditions the equations do not describe the physical processes.

6 Using tabular notation to describe a computational method

A tabular representation is a convenient way to continue the development of requirements documentation after the numerical method for solving the core system of equations is chosen. For example, if we choose to apply finite-difference methods to the problem, we can approximate a temporal derivative as the forward or right difference; we can also approximate a second order spatial derivative with central differences. The finite-difference approximation of the diffusion equation and corresponding initial and boundary conditions from the table DIFF (Figure 3) is given in Figure 4.

DIFFNUM

j = 0	$0<j<N_t-1 \vee j=N_t-1 \wedge$ $0\le\varepsilon\le 1/4 \wedge \varepsilon=k_c\Delta t/\Delta z^2$	$j<0 \vee j>N_t-1 \vee$ $\varepsilon<0 \vee \varepsilon>1/4 \vee$ $\neg$**TURBNUM**

H1

$i = 0$	$c_i^j = c_D$	$c_i^j = c_{i+1}^j - \frac{q_0\Delta z}{k_c}$ $\wedge$ **TURBNUM**	$c_i^j = -1$
$0<j<N_z-1$	$c_i^j = c_D$	$c_i^{j+1} = c_i^j + \frac{k_c\Delta t}{\Delta z^2}\left(c_{i+1}^j - 2c_i^j + c_i^j\right) - \lambda c_i^j$ $\wedge$ **TURBNUM**	$c_i^j = -1$
$i= N_z-1$	$c_i^j = c_D$	$c_i^j = c_D$	$c_i^j = -1$
$i<0 \vee$ $i>N_z-1$	$c_i^j = -1$	$c_i^j = -1$	$c_i^j = -1$

H_2 G

Figure 4: Finite-difference representation of a time-dependant, one-dimensional contaminant diffusion problem.

There j = 0,1,2..,N_t-1, i = 0,1,2..,N_z-1. Further, $\Delta t = P_t / N_t$-1 and $\Delta z = D/ N_z$-1 are the time step and mesh step along the vertical, respectively. The table DIFFNUM refers to another table TURBNUM, which embodies a finite-difference approximations of equations presented in table TURB. In accordance with the relational model of documentation given in Section 3, the non-environmental quantities j, i, N_t, N_z, Δt and Δz, reflecting the details of the chosen numerical method are introduced at this stage by describing the relation NEQ. The stability condition $0 \le \varepsilon \le 0.25$ for the explicit finite–difference representation of the diffusion equation (see, for example, [18]) from the table DIFFNUM (Figure 4) belongs to this relation also. Figure 4 has to be supplemented by additional tables that will describe the finite difference equations for the problem presented in Figure 3. New quantities, which can appear during this process, must be documented in the relation NEQ.

The vertical distribution of strontium 90 concentration calculated using the above approach is presented in Figure 5. This figure adds to the results given in [12]. The computed profiles of the pollutant concentrations reflect the general assumptions and number of simplifications that have been made.

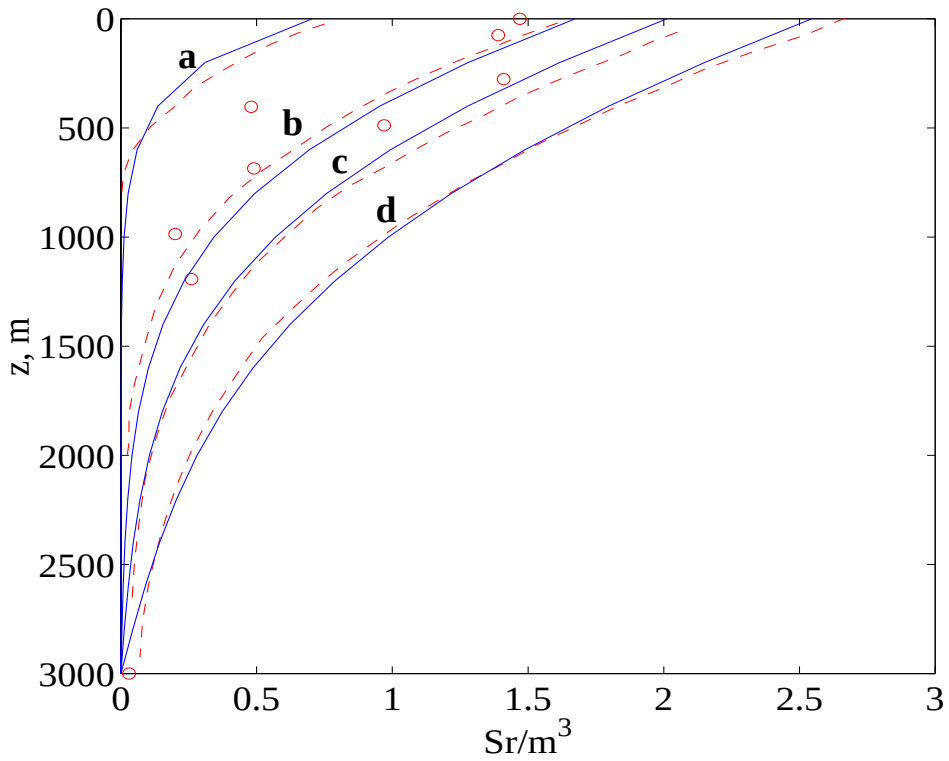

Figure 5: Figure 5. Vertical distribution of strontium-90 concentration in the Atlantic Ocean at the start of (a) 1955, (b) 1958, (c) 1961 and (d) 1964: the solid lines present computed concentrations when average wind speed was 4 m/s, isotope flux from the atmosphere to the ocean was $7.4 \cdot 10^{-10}$ /cm^2 • s, vertical diffusion coefficient was 37 cm^2/s; the dashed lines and circles represent an analytical solution and average field data for 1955 - 1958 as given in Ozmidov [12] correspondingly.

7 Conclusion

The development of software that is based on models of physical phenomena is, unavoidably, an interdisciplinary effort; consequently, to get trustworthy results, it is important that precise documentation be used to communicate between the physical scientists, the engineers, and the software specialists. Such documentation should be used at every stage in the design process. We have illustrated the advantages of using precise formal documentation in the specification, design and review of such software.

In general, the task of such software is to calculate a set of values that are consistent with (a) general physical laws, (b) the detailed characteristics of a specific physical conditions and (c) user preferences about properties such as resolution and accuracy. Engineering projects often require simulation models that approximate the behaviour of physical systems taking many details into account. The physical laws are usually described by sets of equations, that apply within specific space and time boundaries and under specified conditions. The shape of the boundaries, and the conditions at the boundaries, can be quite complicated. The properties of the physical bodies are also complex and may change drastically under certain conditions. In summary, the software must model very complex physical conditions. The requirements for such software are usually described using a combination of natural language and mathematical formulae. Such hybrid descriptions can be unclear and are frequently misunderstood by some members of the software development team. As a result, there are many misunderstandings and these, unavoidably, have a negative influence on the final quality of the software.

We believe that the creation of mathematics-based requirements documents, as an integral part of the software development process, is the only rational way to design trustworthy software for science and engineering. Since the users of such software products must be able to specify the values of certain method-dependant (non-environmental) quantities, the four-variable model introduced in [17] has been extended to a five-variable model to make it possible to describe the meaning and effect of those quantities.

The experience gathered working with tables for real-time problems is sufficient for us to propose that mathematical tables replace more conventional ways of documenting software. This paper illustrates how tabular expressions can be used for documenting the requirements of engineering and scientific modelling software. Moreover, we have illustrated how the same notation can be used when developing the models and methods that will be employed in the software.

In this paper, the five-variable relational model for documenting requirements has been illustrated using a simple example of a time-dependent environmental problem. The approach proposed will allow a user to benefit from the capabilities of the Table Tool System [3]. These tools can assist in several ways such as the production and inspection of tables, and verification of their completeness and consistency. In practical engineering applications with more advanced models, the approach that we have proposed and illustrated can decrease the likelihood of serious errors at the earliest phases of software development process and increase the trustworthiness of the results.

8 Acknowledgements

We are grateful to Professor Spencer Smith for many encouraging and constructive suggestions. We also thank Georgy Kirillen for the computed results in Figure 5.

9 References

1. Boehm, B.W., Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981, ISBN 0-13-822122-7.
2. Hall, A., "Seven myths of formal methods", *IEEE Software*, 7(5), pp. 11-20, 1990.
3. McMaster University Software Engineering Research Group, "Table Tool System Developer's Guide" CRL Report 339, McMaster University, Communications Research Laboratory), TRIO (Telecommunications Research Institute of Ontario), January 1997.
4. Heitmeyer, C., Bull, A., Gasarch, C., Labaw, B.G., "SCR: A Toolset for Specifying and Analysing Requirements", *Proceedings of the 9th Annual Conference On Computer Assurance (COMPASS'95)*, Gaitherburg, MD, 1995, pp. 109-122
5. Heitmeyer, C., Lynch, N., "Formal Verification of Real-time Systems Using Timed Automata", in *Formal Methods for Real-Time Computing*, (Heitmeyer, C., Mandrioli, D. Eds.), J Willey and Sons, Chichester, New-York, Brishane, 1996, pp.83-106
6. Heninger, K.L., Kallander, J., Parnas, D.L., Shore, J.E., "Software Requirements for the A-7 Aircraft", *NRL Memorandum Report 3876*, United States Navel Research Laboratory, Washington DC, November 1978
7. Hester, S.D., Parnas, D.L., Utter, D.F., "Using Documentation as a Software Design Medium", *Bell System Tech. J.* **60(8),** pp. 1941-1977
8. Janicki, R., "Towards a Formal Semantics of Parnas Tables", *Proceedings of the 17th International Conference on Software Engineering (ICSE'95)*, IEEE Computer Society, Seattle, WA, 1995, pp. 231-240.
9. Janicki, R., Khedri, R., "On a Formal Semantics of Tabular Expressions, *SERG Report.379*, McMaster University, September 1999.
10. Janicki, R., Parnas, D.L., Zucker, J., "Tabular Representations in Relational Documents", in *Relational Methods in Computer science*, Brink, C., Kahl, W., Schmidt, G. (Eds), Springer-Verlag, 1995, pp. 184-196.
11. Kreyman K., Parnas, D.L., Qiao, S., "Inspection Procedures for Critical Software That Model Physical Phenomena", *CRL Report No. 368*, McMaster University, Hamilton, Ontario, Canada, February 1999.
12. Ozmidov, R.L. Diffusion of Contaminant in the Ocean, Kluwer Academy Publisher, Dordrecht, Boston, 1990.
13. Parnas, D.L., "Tabular Representation of Relations", *CRL Report 260*, McMaster University, CRL, Telecommunications Research Institute of Ontario (TRIO), 1992.
14. Parnas, D.L., "Predicate Logic for Software Engineering", *IEEE Trans. Software Eng*. **19 (9),** 1993, pp. 856-862.
15. Parnas, D.L., Asmis, G.J.K, Madey, J., "Assessment of Safety Critical Software in Nuclear Power Plants", *Nuclear Safety*, **32 (2),** 1991, pp. 189-198.
16. Parnas, D.L., Clements, P.C., "A Rational Design Process: How and Why to Fake It", *IEEE Trans. Software Eng.* **SE-11**, 1986, pp. 251-257.
17. Parnas, D.L., Madey, J., "Functional Documentation for Computer Systems Engineering", in *Science and Computer Programming*, (Elsevier) **25 (1)**, October 1995, pp. 41-61
18. Ramming, H.-G, Kowalik, Z., "Numerical Modelling of Marine Hydrodynamics. Application

to Dynamic Physical Processes", Elsevier Scientific Publishing Company, Amsterdam, Oxford, New–York, 1980

19. Rojiani, K.B., Programming in C with numerical methods for engineers, Prentice Hall, Inc. 1996,
20. van Schouwen, A.J., Parnas, D.L., Madey, J., "Documentation of Requirements for Computer Systems", *Proceedings of'93 IEEE International Symposium on Requirements Engineering*, San Diego, CA, 4 - 6 January, 1993, pp. 198 - 207.
21. Zou, Y. Application of the Four Variable Model to a CAD System for Microwave Communication Devices, *SERG Report 387*, McMaster University, Dept. of Computing and Software, June 2000, 168 pgs.
22. Zucker, J., "Transformations of Normal and Inverted Function Tables", *Formal Aspects of Programming*, **8**, 1996, pp. 679-705.