# Inspection Procedures for Critical Programs that Model Physical Phenomena

Konstantin Kreyman, David Lorge Parnas, Sanzheng Qiao
DEPARTMENT OF COMPUTING AND SOFTWARE
Faculty of Engineering
McMaster University

## Abstract

This paper addresses the problem of assuring the accuracy and trustworthiness of computer programs that are based on models of physical phenomena. It begins by explaining why such programs can be critical to safety, health, and property. After a quick review of what is known about software inspection, it presents a classification of the possible sources of error in these programs. Subsequently, the paper outlines an inspection procedure that would find these errors. The final section of the paper illustrates the concepts using a simple example, a program that might be used to predict contaminant diffusion.

## I When is software critical?

The software that controls a nuclear plant, chemical plant, automobile, or airplane is obviously critical to the safety of the equipment and people near it. Clearly, software used for monitoring ongoing processes, e.g. for observing the level of contaminants, can also be relevant to safety. It is not as obvious, but it is equally true, that software used to model complex physical phenomena during the design or analysis of products and plants may also be critical for both safety and property protection. If the computer programs, which are used to predict what will happen after the plant or product is put into service, produce results that are not correct the consequence could be a failure that leads to loss-of-life and/or environmental damage.

Software that controls systems in “real-time” has long been a concern in the software engineering research community; there has been a great deal of research work on the inspection or formal verification of that software [e.g. 23, 24]. However, the software used in the design and analysis of products and installations during the design phase, has not received the same attention. Software is often used in many critical analyses, such as

- to predict the effect of explosions,
- to estimate radioactive contaminant diffusion,
- to calculate the effects of chemical transport in atmosphere and water bodies, and
- to estimate the dispersal of heated water into natural reservoirs.

It is often assumed that the engineers who use such scientific software can check the results before they use them. This assumption is false. If it takes a computer to perform the analysis, it will be difficult for any engineer or scientist to perform anything more than crude limiting case checks on the results. Further, we all get into the (bad) habit of trusting the results our computers give us. Subtle, but important, errors may remain undetected. Systematic methods of inspecting and certifying the behaviour of software for these applications would be very useful.

## II Software quality assurance: the state of the art

Few people in the software industry like to talk to the public about the problem, but almost all the software in use today contains serious errors, euphemistically called “bugs”. Errors are always present when software is first written, but many are removed during inspection and testing. The bugs that do not get corrected are the errors that manifest themselves only rarely or that result in errors that are not noticed. It seems that if software works “most of the time”, people will buy it and use it. Unfortunately the errors that remain can lead to expensive errors as well as to extensive loss of productivity.

The software industry does not seem to know how to get rid of these bugs. The obvious solution is to treat programs as mathematical objects and to prove them correct in the same way that we prove mathematical theorems to be correct. This idea, first advanced in the 50s and 60s [e.g. 6], is a sound mathematical basis for verifying the correctness of programs. However, in spite of more than 30 years of research, most researchers recognise that these techniques are still not ready for widespread use. Proofs can be prepared by highly educated computer scientists working on carefully crafted programs that are written using specially designed programming languages. Even in those cases, the process is very expensive and bugs escape detection. It often takes more effort to prove (formally) that a program is correct than to write the program. More seriously, that effort does not provide a complete check on the program. All of the popular proof techniques assume perfect arithmetic and ignore exception handling mechanisms. All of these methods assume that we have a specification in the form of mathematical assertions about the program and consequently, cannot deal with programs that have incorrect specifications or no specifications at all. Moreover, the notation used for writing the specifications, while simple in itself, results in complex mathematical expressions that cannot be read by the subject matter specialists who really know what the program will do. The expressions are also sufficiently complex that it is easy to overlook cases that are not described properly.

A more promising approach is systematic inspection. While not as rigorous as a formal mathematical proof, a good inspection process will apply the “divide and conquer” principle to reduce the job of inspecting a large and complex program to a number of small, relatively simple, inspection tasks.

The most popular effective approaches to software inspection derive from the work of Fagan while at IBM [3, 4]. These informal methods have been found to be good ways to find errors, but not to eliminate errors. It is common for errors to remain undetected after informal inspections. Such inspections can also be very expensive because they rely on large-group program-reading sessions.

## III Software Inspection Based on Program Function Tables

A more effective form of inspection has been developed by combining the ideas of Fagan with those of H.D. Mills [17] and Parnas, Madey and Iglewski [21, 25]. This method was developed for application in the inspection of safety critical control programs [23].

Mills showed how conventional (i.e. classical) mathematics could be used to prepare precise summaries of a program's behaviour [17]. Parnas, Madey and Iglewski showed how the readability of the summaries could be improved by the use of tabular notation [10] and structured into a set of displays [21]. A pictorial version of displays has also been proposed in [22]. The display based inspection method was first used in the inspection of nuclear plant software as described in [23, 24]. It has also been applied to other control software in industrial short courses. In most cases, display-based inspection reveals serious errors that were missed by other forms of review and testing.

In the display-based, or program-function-table based method, each program is described by a function or relation that maps from “start states” to “final states” as proposed by Mills. The functional descriptions are organised into a set of displays, each one of which can be inspected without looking at another display, as described in [21]. We have found that by using tabular mathematical expressions, the information can be presented in an intuitive and useful way.

Four teams should be involved in the inspection process. The first team produces a tabular representation of the requirements, i.e. a statement of what the program *should* do. The second team, comprising people who do not know the application area, produces a description of what the program actually does. A third team, compares the two mathematical descriptions in an attempt to show that they are equivalent. The attempt usually fails but reveals a number of discrepancies. Often these discrepancies are places where the program is right and the requirements were incorrectly stated, but there are usually some unsuspected errors in the programs found as well. The fourth team audits the whole process. More detailed discussion of the process can be found in [23, 24].

The work of the second team is based on a hierarchical decomposition of the program that allows the program to be examined in small parts. The use of the tabular notation allows the inspection and comparison to proceed systematically on a case-by-case, variable-by-variable basis. The result is a "divide and conquer" approach that reduces a complex task to a set of much simpler tasks.

Two sets of tools, one the product of academic research [14], one for industrial use [15,16], are being developed to support this inspection process. It is believed that such tools can greatly reduce the cost, and consequently enhance the applicability, of the inspection process.

We believe that the program-function based inspection method is the most effective practical approach to certification of programs. However, up to now all applications of this method have been control programs, not programs that model physical phenomena. In section IV, we explain a variety of types of errors that can be found in such programs. Section V provides a very brief summary of earlier work on such problems. Section VI discusses ways to improve the inspection process so that it is more suited for such programs. Section VII illustrates these errors using a simple example.

## IV Possible Sources of Error in Programs that Model Physical Phenomena

We have identified six major types of errors that may be found in programs that model physical phenomena. These are:

1. The scientists may have chosen models of the physical phenomena that are inappropriate for the circumstances. Of the many models that can be used to predict the behaviour of a physical system, some are simple but not very accurate while other, more complex, models may be more accurate but computationally intractable. A model that ignores one or more physical facts may be quite adequate in some situations but give misleading results under other conditions. When models are "buried" in a computer program, their weaknesses or inadequacies may not be noticed.
2. The program may not correctly embody the model(s) chosen by the scientists. Program design errors are very common and can be very subtle. Often, those who do the programming do not fully understand the model. Similarly, the scientists who chose the model may not be skilled in programming and may not notice that the programme is not a faithful implementation of their model.
3. The algorithm used for calculating numerical results may be unstable and compute values that are very inaccurate for some inputs. Because digital computers can only approximate real numbers, numerical errors in engineering and scientific calculations are inevitable. Even simple, algorithms can appear sound but be unstable. Some algorithms allow errors to accumulate until they severely distort the results. Consequently, two algorithms that are logically equivalent[1]can compute different numerical values when implemented with finite arithmetic.
4. Logical errors can arise because the programmer has overlooked certain special cases. For example, a programmer may write a program that discards "outliers" (values that are far from a calculated mean) but forget that, in some cases, this would eliminate all but a few data points from consideration. Another example: a programmer may deal with the case where a measured value is

[1] We consider two programs logically equivalent if they would compute the same results with perfect arithmetic.

above a limit and the case where it is below the limit, but forget the case where the value is exactly at the limit. Loops are a frequent source of logical errors as they may terminate one step too soon or one step too late.

5. Errors may be introduced into computer models of physical phenomena when several programs, each of which is an accurate model of an individual process, are combined. Sometimes, the data structures used for the component modules are incompatible (e.g. the mesh size may differ). Often two programs may have different accuracy standards or data may be stored using different units or scaling factors. Logical errors may be introduced if two models use the same variables in incompatible ways. When two processes work on different time scales, the combined program may evaluate some expressions either too frequently or too infrequently[2]. Often too, when two models are combined, quantities that were considered constants in one model may become variable. If a program has been “optimised” on the assumption that some quantity was a constant, the results of the combined program may be incorrect. Because, logical, numerical, and modelling errors may be introduced when programs are combined, it is important that the composite program be subjected to an especially careful inspection, but this is rarely done because people assume that the combination of two reliable programs will also be reliable.
6. Errors may be introduced by a multi-phase approach to computation. In order to obtain a program that is an accurate model of a physical system, it is sometimes necessary to increase the number of variables and use complicated sets of mathematical equations with detailed initial and boundary conditions. However, such models can be slow to converge unless they are initialised with data that are approximations of the final values. Consequently, in some cases, it is necessary to use both crude and detailed models in the same program [11, 26]. The crude (first) phase is used to get an approximate initial solution; the detailed models are used in the second phase to refine the initial solution to a more accurate one. This multi-model/multi-step approach can greatly simplify the models and the program design process, but it introduces new problems because one must now worry about the interface between the two phases. For example, the boundaries chosen may not be suitable because the cruder models don’t predict conditions accurately along a boundary. An inspection process must both (1) determine if the decomposition was valid and (2) determine if the first phase always terminates with results that are suitable for the second phase.

## V What is known about these problems?

In the past logical errors, modelling errors, and numerical errors have been studied separately. Logical Errors have been dealt with successfully by the process described in [23, 24], but this work does not address numerical or modelling errors. There is a great deal of work on numerical errors [e.g. 1, 8, 13, 28, 29], but this work does not describe a systematic inspection procedure. Specialised papers such as [2, 5, 11] discuss modelling issues, but there is no unified approach that deals with the complex problems that arise when the errors discussed above are present.

The high probability of errors in programs that model complicated physical phenomena has led scientists to significantly simplify models by neglecting some (possibly important) physical features. For example, such an approach is used to model environmental hydrodynamic processes [12, 30]. With the growing dependence on software to calculate results which, if incorrect, may endanger the public or cause serious damage to equipment, we believe that it is imperative that we develop new inspection procedures and tools that can assist software engineers to work together with scientists and mathematicians to make the program inspection process more effective.

[2] When expressions are evaluated too frequently, unnecessary numerical error may be introduced; if they are evaluated too rarely, the results may only be crude approximations of what is needed.

Research in this area must be interdisciplinary; it requires an understanding of the physical processes that is possessed primarily by physical scientists, an understanding of mathematics and the numerical behaviour of programs, that is possessed by mathematicians, and an understanding of programming that is primarily possessed by computer scientists.

## VI An Generalised Inspection Approach

We propose an inspection procedure for programs that predict the effects of critical physical phenomena as well as programs that are used in monitoring physical systems. The following procedure is derived from the function-table based inspection process described in section III but extended so that it deals with all of the problems discussed in section IV.

0. During the model development phase, the designers should have developed a set of equations that they consider to be a good model of the physical processes involved. This task is often done informally and the results are often not well documented. A precisely documented statement of the basic model is essential to an effective inspection process. If such documentation is not available, a precisely documented specification for the software will have to be produced before the inspection process can proceed to phase 1.
1. The first step in the inspection is the decomposition of long program into parts small enough to that can be reviewed to determine the mathematical functions that they implement. In this phase we can use the method described in [23, 24]. If a program is well-structured, it will naturally decompose into smaller sections that correspond to individual computational models and data representations. With well structured programs, the inspection process is always simpler, faster, and more reliable. If a program is badly structured, it may be advisable to restructure it before inspecting it.
2. For each of the small parts, inspect for numerical errors. We may be able to use a specially developed checker program that will determine whether or not the output information is within certain error bounds depending on the algorithm sensitivity, stability and rounding errors as described in (for example) [27]. We also believe that interval arithmetic techniques [7,18] can be used to develop a tabular description that states the range of possible values explicitly. If the result is unacceptable, the stability of the algorithms must be reviewed. It is assumed that the checker will be essentially different from the program, simpler, faster and include self-correction features eliminating errors. After inspecting each of the small parts separately, the numerical behaviour of the whole program may be reviewed.
3. Using the techniques described in [24], combine the precise descriptions obtained in step 2 to obtain a precise description of the model of physical phenomenon including initial and boundary conditions that is actually embodied by the program. The results of both this phase, and the previous phase will be expressed in the form of tabular mathematical expressions [10].
4. Compare the descriptions obtained in phase 0 with those obtained in phase 3 using a systematic (column by column, row by row) procedure that attempts to show that the two tabular expressions are representations of the same function. The availability of two precise descriptions, one describing the desired model and one describing the actual program is a major advantage of this procedure. It is not possible to make precise comparisons of an informally described model and a complex program.
5. Verify that boundaries and phases have been chosen properly. When a model is a combination of smaller models, each dealing with a clearly defined subset of the processes being studied, it is possible that the boundaries are chosen inappropriately. It may happen that because of the inappropriately chosen boundary, the earlier phases of the computation do not compute appropriate initial boundary conditions for use in the later phases.

It is likely that the result of any effective inspection process will be a decision to modify the program. If modification is necessary, the program developers will be able to use the two detailed tabular descriptions to guide the modification. They will show the difference between the program and desired model of the real physical phenomenon. The use of the inspection approach described above will increase the ability of both the program developers and the scientists to assure the correctness of the program's results. For the first time there will be a sound basis for the modification and certification of programs that embody models of critical physical phenomena.

The proposed inspection procedure permits scientists who investigate physical processes to play an active role in the program development process throughout the project and help to eliminate the serious gap between program developers and physical scientists that exists today.

## VII An illustrative example: contaminant diffusion

This section presents an example of a situation in which computer modelling of complex physical phenomena can be used to solve a practical engineering problem. The problem is to predict the diffusion of contaminants in a body of water in the event of an accidental spill. Section 1 describes the physical models that are proposed and also presents PASCAL programs that are based on those models. Section 2 illustrates problems discussed in this report, i.e., logic, numerical and modeling errors in computer programs.

We have chosen this illustration in recognition of the growing role of software in environmental applications. With the growing public awareness of the damage that can be done by new construction, both the owners and regulatory bodies are depending upon computer models to demonstrate that the effects of accidents will be limited and that the environment can recover from accidental damage. Like all natural phenomena, the environmental phenomena discussed below are complex. To illustrate the issues discussed above, we have simplified the mechanisms and reduced the number of variables in order to keep the examples clear.

### 1 Two simple models of physical phenomena

The diffusion of contaminants in water bodies and atmosphere is a major factor determining the distribution of both natural and man-made substances introduced into water and air. Prediction of contaminant diffusion in natural environments is a difficult task. Based on experimental and mathematical studies on the turbulent diffusion in natural media, scientists have created computer models suitable for practical applications [2, 20]. Here, we consider radial turbulent diffusion in two dimensions of a patch of pollutant initially concentrated at one point of a water body. For simplicity, we assume that the turbulence is two-dimensionally isotropic. We further assume that all of the contaminant is injected into the water at the same moment. Let *S* be the mass of contaminant that is dispersed in the water (per unit of surface area), *t* the time since the contaminant was inserted, and *r* the distance from the point at which the commandant was inserted; the general diffusion equation is then:

$$\frac{\partial S}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left[K(r) r \frac{\partial S}{\partial r}\right] \quad , \tag{1}$$

where *K*(*r*) is known as the coefficient of turbulent diffusion. Under realistic assumptions, *K*(*r*) is given by

$$K(r) = c_1 \varepsilon^{1/3} r^{4/3}$$

where ε is the turbulent energy dissipation rate and $c_1$ is a universal, non-dimensional, constant. From this, we get the solution to the diffusion equation (1) [20]:

$$S(r,t) = \frac{S_0}{6\pi(kt)^3}\exp(-r^{2/3}/(kt)) \tag{2}$$

where $k = 4c_1\varepsilon^{1/3}/9$ and $S_o$ is the mass of the contaminant source. Formula (2) can be used to calculate the concentration of contaminant if the dissipation rate ε is known. The value of ε depends on the dynamic conditions of the medium. Its calculation is another rather challenging task in environmental hydrodynamics. However, it is often possible to use estimations of dissipation rate that were obtained experimentally. For example, according to [19], an empirically determined ε = 0.3 $cm^2/s^3$ can be regarded as adequate for the surface layer of natural water body.

The model described above has been used to develop the following PASCAL program:

```
const
     pi = 3.1415926535898;
     C1 = 1.19;   {universal constant}
     E = 0.3;       {turbulent energy dissipation rate,}

   function Concentration(S_0, r, t: real): real;
     {S_0: source intensity}
     {r: distance from the origin }
     {t: time}
   var
     u: real;
   begin
     u := 4*t*C1*power(E, 1/3)/9;
     Concentration := (S_0/(6*pi*u*u*u))*exp(-power(r, 2/3)/u);
   end; {Concentration}
```

Figure 1. Program Concentration

Note that in the above program we assume a function:

```
function power(x, y: real): real
```

which computes $x^y$ is available.

Now, we consider the computation of the intensity of the contaminant source, i.e., $S_o$ in (2). For simplicity, we assume that, because of the design of the device that discharges the contaminant, the value of the point source $S_o$ depends on the water temperature $T$ as follows:

$$S_o = S_d(T_d/T) \ , \tag{3}$$

where $T_d$ is the temperature at which the intensity of the contaminant source reaches the maximum $S_d$.

The above model is implemented by the following PASCAL program:

```
var
    S_d,       {maximum value of contaminant source}
    T_d: real;  {temperature when S_d is reached}
function SourceInten(T: integer): real;
    { T: temperature }
begin
    SourceInten := S_d*(T_d/T);
end; {SourceInten}
```

Figure 2. Program SourceInten

A program is also needed to determine the value of the temperature T in (3), the parameter passed to the above function. We consider the variation of temperature in a water reservoir with constant depth, *D*. Assuming horizontal homogeneity of the temperature field in the reservoir, the mean temperature, *T*, of the water body must satisfy the following "heat budget" equation

$$\frac{dT}{dt} = (Q_s - Q_D)/D, \tag{4}$$

where $Q_D$ is the kinematic heat flux, i.e. the heat flux divided by the product of the density and the specific heat capacity of the water, through the reservoir bottom and $Q_s$ is the kinematic heat flux through the free water surface. If we assume that in warm seasons the heat flux through the bottom comparing with the heat flux through the free water surface is so small that it can be neglected, then, denoting the initial temperature as $T_0$, we get the solution to (4):

$$T = T_0 + \frac{Q_s t}{D}. \tag{5}$$

The heat flux $Q_s$ is positive when it is directed from the atmosphere into the water body.

The following PASCAL program implements this model of temperature regime of a homogeneous water body.

```
var
    T_0,        {initial temperature}
    Q,          {heat flux}
    D: real;    {depth}
  function Temperature(t: integer): real;
    {t: time}
  begin
    Temperature := T_0 + ((Q*t)/D);
  end; {Temperature}
```

Figure 3. Program Temperature

Combining the above three programs, we obtain the following (composite) program:

```
Program Composite(input, output);
 var
    t: integer; {time}
    r, {distance from the origin}
    S: real; {mass per unit surface area}
 begin
    write('Enter time t: ');
    readln(t);
    write('Enter distance r: ');
    readln(r);

    S := Concentration(SourceInten(Temperature(t)), r, t);
 end; {Composite}
```

Figure 4. Program Composite

## 2 Problems in the programs

Below we discuss some errors in the above PASCAL programs to illustrate our list of possible sources of errors in programs that model physical phenomena.

### 2.1 Logic error

In the function Concentration, the programmer forgot the case where $t = 0$. This kind of logical errors can be easily revealed by determining the tabular description of the function as in [10, 23].

### 2.2 Errors introduced when several programs are combined

A less obvious logic error occurs when the programs Temperature and SourceInten are combined. In particular, the mean temperature returned by Temperature is of type real. This is correct as a separate physical model. However, SourceInten requires that the temperature be restricted to values not less than $T_d$. In our example, we set $T_d = 1.0$. To ensure that $T \geq T_d$ and consequently $S_o \leq S_d$, in the program SourceInten in Figure 2 we declare the water temperature $T$ (> 0) as an integer. Thus, it is necessary to convert the value (temperature) returned by the program Temperature into data type integer before it is passed to SourceInten. This kind of data incompatibility caused by the composition of programs can be hard to detect. Sometimes, such incorrect results may seem acceptable, especially for programmers who are unfamiliar with the physics of the phenomenon. Figure 5 shows the difference between two computations, one with the data type conversion (curve a) and one without (curve b). While the shapes of the curves are similar, i.e., the nature of the dependence of the concentration on time at the beginning stage of the diffusion process is the same, but the values of the contaminant concentrations that resulted from the two calculations are significantly different.

Furthermore, the time $t$ in Concentration and the time $t$ in Temperature have different reference points. The former is the time since the introduction of admixture in the diffusion model, whereas the later is the time since the beginning of heating in the temperature model.

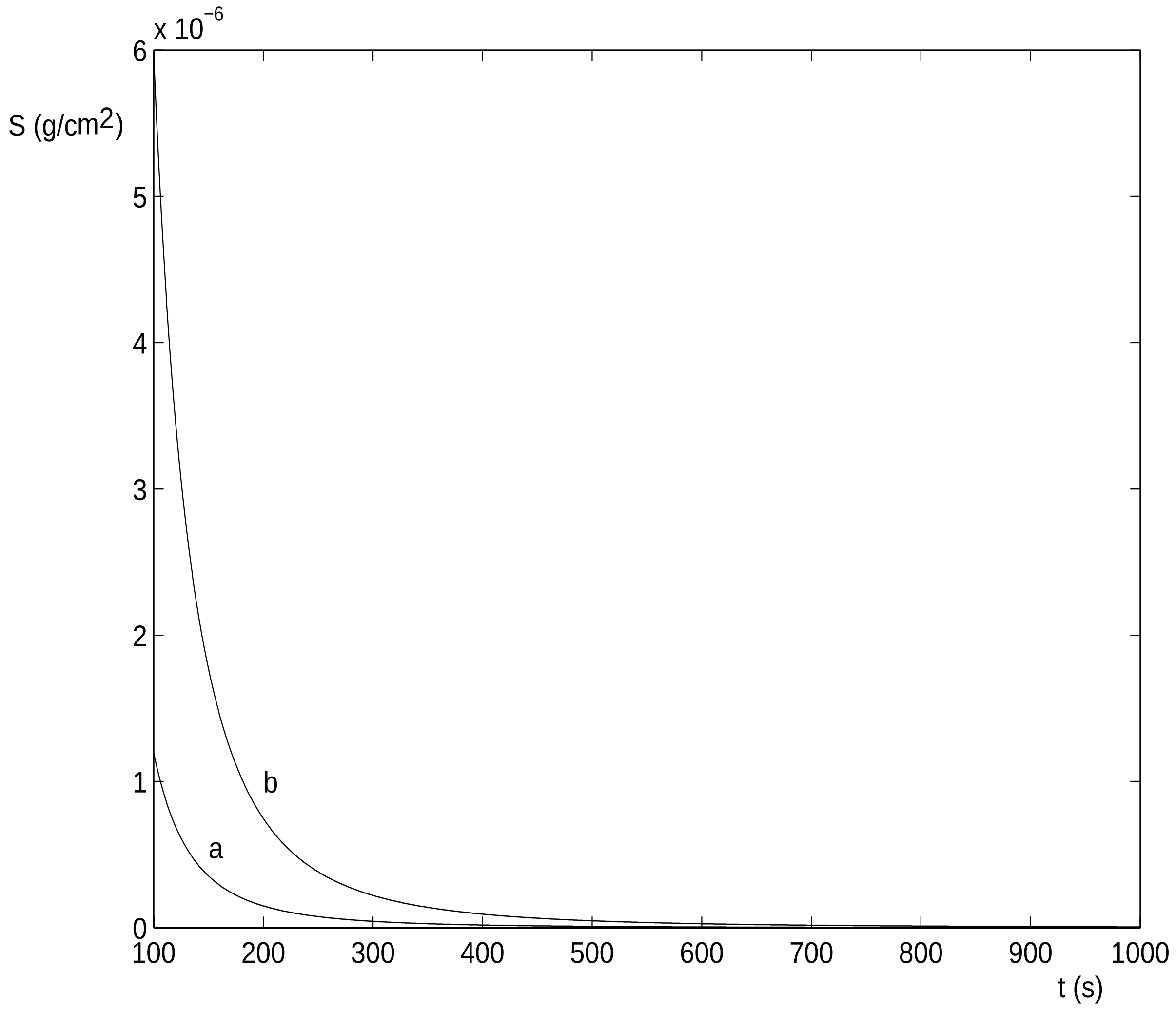

Figure 5. Time dependence of the *S* at the origin of the coordinates ($r = 0$) calculated by the program Composite where $T_o = 0.1^o$ C, $Q = 0.01$ $K \times cm/s$, $D = 1000$ cm. The admixture was introduced into the water body from an instantaneous point source when the temperature *T* reached $1.0^o$ C (in case a) and $0.2^o$ C (in case b) after heating periods of $9 \times 10^4$ *s* (in case a) and $10^4$ *s* (in case b). In case (a), the temperature *T* computed by "Temperature" was converted to integer while in case (b), *T* was not converted.

### 2.3 Modeling error

The discovery of the logic error due to $t = 0$ in section 3.1 warns us to consider the case when *t* is close to zero. From (2), we have

$$\lim_{t \to \infty} S(r,t) = \lim_{r \to \infty} S(r,t) = 0 \qquad (6)$$

Also, from (2), for any fixed $t > 0$, as *r* tends to 0, $S(r,t)$ approaches $O(t^{-3})$, which can be large when *t* is small. So, there must be a maximum when both *r* and *t* are small. Specifically, it can be verified that when $t = 3r^{2/3}/(4c_1)$, $S(r,t)$ reaches its maximum $9S_o e^{-3} / (2\pi r^2)$. However, this does not accurately reflect the physics of the beginning stage of the turbulent diffusion from a instantaneous point contaminant source. In other words, (2) is inadequate for small values of *r* and *t*. The proper ranges for *r* and *t* should have been determined based on theory and confirmed by experiments. When this is done, one gets the results shown in Figure 6.

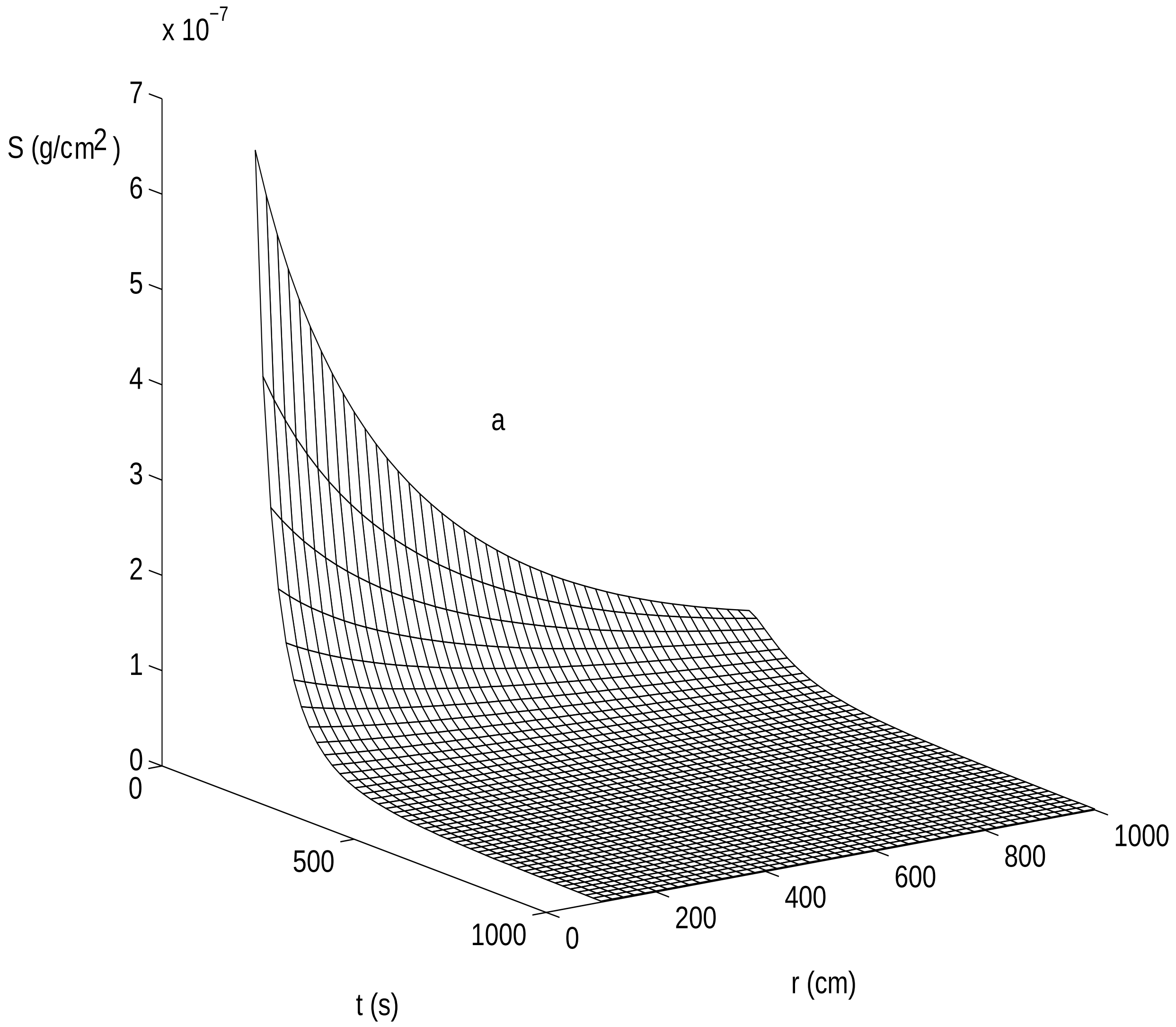

Figure 6a. Plot of *S(r,t)* computed by the program Concentration where $S_0 = 1.0\ g$. In case (a), $100 \le t \le 1000\ s$ and $100 \le r \le 1000\ cm$.

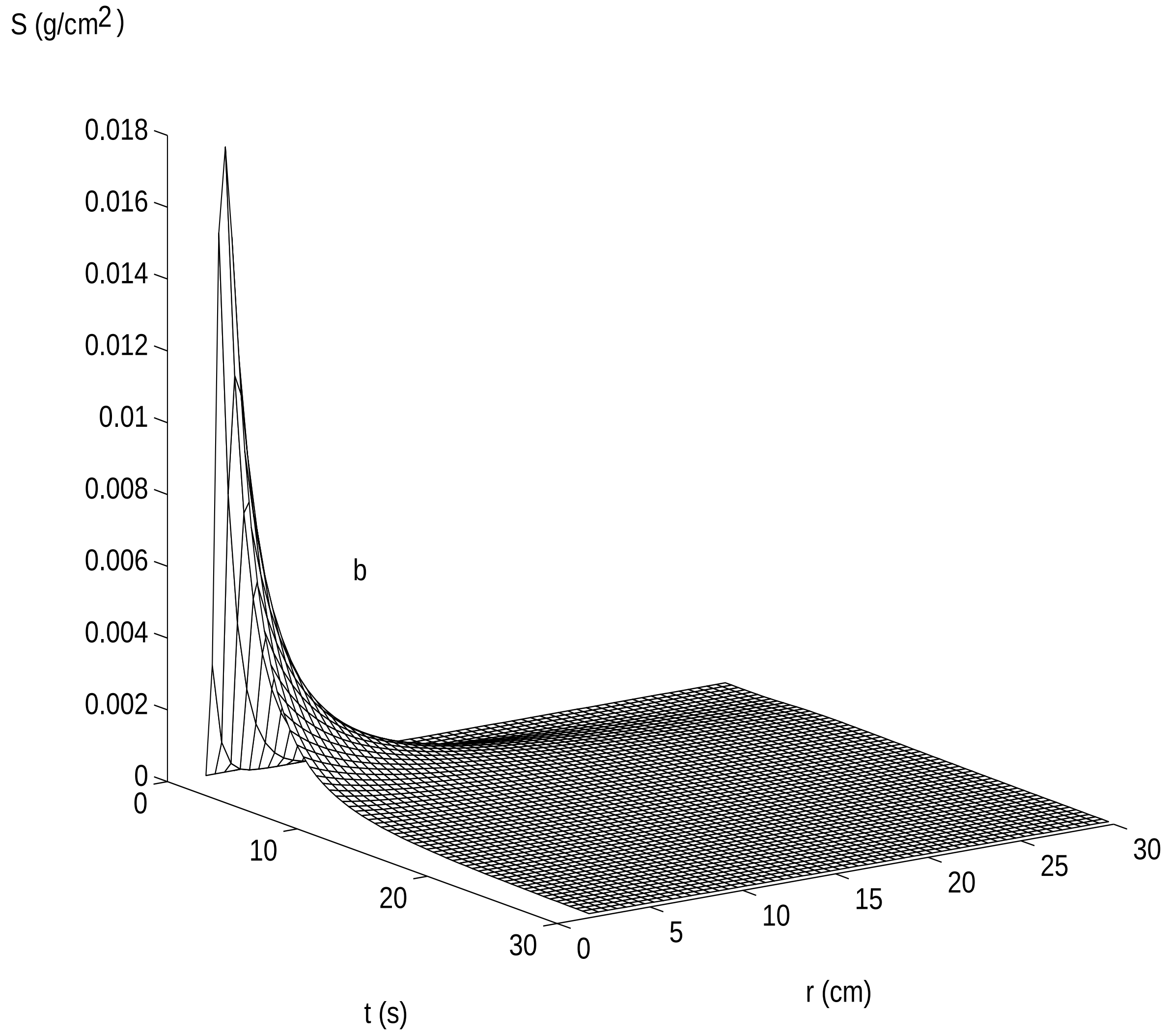

Figure 6b. In case (b), $0.1 \le t \le 30$ *s* and $2 \le r \le 30$ *cm*.

## 2.4 Numerical error

Program Concentration may cause an unnecessary underflow exception. Consider a system with IEEE floating-point standard single precision without gradual underflow [9]. The smallest positive number in such system is $2^{-126} \approx 10^{-38}$. When Concentration is executed, temporary variables

$$u = 4tc_1\,\varepsilon^{1/3} / 9, \quad t_1 = 6\pi u^3 \quad \text{and} \quad t_2 = \exp(—r^{2/3} / u)$$

are calculated, then $(S_0 / t_1)t_2$ is computed for the return value. When $S_0 = 1.0$ and $t = 0.3904$, we get $u=4tc_1\,\varepsilon^{1/3} / 9 = 0.1382$. If $r = 42.61$, then $r^{2/3} = 12.20$ and the evaluation of $t_2 = \exp(—r^{2/3} / u)$, which is about $4.7 \times 10^{-39}$, causes underflow exception. However, the final result $(S_0 / t_1)t_2$, which is about $9.5 \times 10^{-38}$, does not underflow. In other words, the underflow exception caused by the evaluation of $\exp(—r^{2/3} / u)$ is unnecessary. To circumvent this problem, we rewrite (2), using $u^3 = e^{3 \ln u}$,

$$S(r,t) = (S_0 / (6\pi))\exp(—(r^{2/3} / (kt) + 3 \ln (kt))).$$

The evaluation of the above expression gives $9.5 \times 10^{-38}$ without underflow exception.

## 2.5 Inappropriate model

Finally, we use this example to elaborate the source of error presented in section IV (paragraph 1), "The scientists may have chosen a model of the physical phenomena that is inappropriate for the circumstances". In this example, the model (2) implies that the contamination exists everywhere immediately. In other words, this model predicts that the contaminant spreads infinitely far with infinite speed. Obviously, this does not accurately describe the physical phenomena. If a very small contamination can be neglected, (2) provides an acceptable and simple model. However, when even a trace of contamination is critical, model (2) may be too crude and cause a false alarm. A more appropriate model should be chosen.

## 3 Conclusion

In this paper we have described and illustrated the classes of mistakes that can be found in computer programs that are used to solve engineering tasks by modelling physical phenomena. Each of the programs is simple, straightforward, and appears to be correct. However, we have identified several types of relatively subtle problems can lead to incorrect results with potentially dangerous consequences.

The fact that even such short and simple programs can contain such errors shows that it is necessary, in fact urgent, that we develop systematic practical inspection procedures for such programs. The program-function-table based inspection procedures developed for safety-critical control programs are a good start, but they do not deal with the additional problems that are encountered when physical phenomena must be modelled to predict the effects of future events accurately. In practical applications, with more complicated models and programs, the likelihood of various errors is much higher and the necessity of systematic, mathematically based, inspection procedures is even more important.

## VIII Acknowledgments

We appreciate the efforts of Nitish Mukhi in preparing the illustrations in this paper.

## IX References

[1] Ar, S., Blum, M., Codenotti, B., Gemmell, P., "Checking Approximate Computations Over the Reals", in *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, San Diego, CA, May 16-18, ACM, New York, 1993, pp. 786-795.

[2] Chapra, S.C., Rackhow, K.H., "Engineering Approaches for Lake Management, V.2, Mechanistic modeling", Boston, London, Butterworth Publishers, 1983.

[3] Fagan, M.E., "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, No. 3. 1976, pp. 184-211.

[4] Fagan, M.E. "Advances in Software Inspections", IE*EE Trans. Software Engineering*, July 1986, pp. 744-751.

[5] Fedorovich, E.E., Golosov, S.D., Kreiman, K.D., Mironov, D.V., Shabalova, M.V., Terzhevik, Yu, A., Zilitinkevich, S.S., "Modeling Air-lake Interaction, Physical Background", S.S. Zilitinkevich Editor:, Springer-Verlag, 1991.

[6] Floyd, R.W., "Assigning Meanings To Programs", *Proc. of the Symposium of Applied Mathematics*, vol. 19, 1968. Also In: Schwartz, J.T. (ed), *Mathematical Aspects of Computer Science*, American Mathematical Society, 1967, pp. 19-32.

[7] Hammer, R., Hocks M., Kulisch, U., Ratz D., “Numerical Toolbox for Verified Computing I. Basic Numerical Problems: Theory, Algorithms, and Pascal-XSC Programs. Springer-Verlag, Berlin, 1993. ISBN 3-540-57118-3

[8] Higham, N.J., “Accuracy and Stability of Numerical Algorithms”, *SIAM*, 1996.

[9] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985. Institute of Electrical and Electronic Engineers, New York, 1985. Reprinted in *SIGPLAN Notices*, 22(2):9-25, 1987.

[10] Janicki, R., Parnas, D.L., Zucker, J., “Tabular Representations in Relational Documents”, in *Relational Methods in Computer Science*, Chapter 12, Ed. C. Brink and G. Schmidt. Springer Verlag, 1997, pp. 184 - 196.

[11] Kreyman, K.D., Oganesyan, L.A., Kirillin, G.B., Zverev, I.S., “Simulation of Hydrodynamic Processes in Coastal Water of Big Lake”. *Abstracts of the Second International Lake Ladoga Symposium. The Largest Lake in Europe and Its Environment*, August 1996, Joensuu, Finland, p. 30.

[12] Kreyman, K.D., Oganesyan, L.A., Kochkov, N.M., Terzhevik, Yu, A., Golosov S.D., “Modelling the Main Thermodynamic Mechanisms of Lake Ladoga”. *Abstracts of the First International Lake Ladoga Symposium*. University of Joensuu, Finland, Publications of Karelian Institute, No.112, 1995, p.68.

[13] Luk, F.T., Qiao, S., “Analysis of a Linearly Constrained Least Squares Algorithm for Adaptive Beamforming, Integration”, *The VLSI Journal* Vol. 16, No. 3, Dec. 1993, pp. 235-245.

[14] Matias, E.D., “Software Engineering, Standards and Methods (SESM): Tools Overview”, AECL Report COG-98-013-I (Rev. 0), March 1998.

[15] 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.

[16] McMaster University Software Engineering Research Group, Appendices to the Table Tool System Developer’s Guide, CRL Report 340, McMaster University, CRL (Communications Research Laboratory), TRIO (Telecommunications Research Institute of Ontario), Jan. 1997.

[17] Mills, H.D. “The New Math of Computer Programming”. *Communications of the ACM*, vol. 18, No. 1, January 1975, pp. 43-48.

[18] Nickel, K., “Interval-analysis”, in *The State of the Art in Numerical Analysis*, David A.H. Jacobs, editor, Academic Press, London, 1977, pp. 193-225

[19] Nihoul, J.C.J. (Editor), “Modelling of Marine Systems”. Amsterdam, New York, Elsevier Scientific Pub. Co., 1975.

[20] Ozmidov, R.V., “Diffusion of Contaminants in the Ocean”. Dordrecht, Boston, Kluwer Academy Publisher, 1990.

[21] Parnas, D.L., Madey, J., IglewskI, M., “Precise Documentation of Well-Structured Programs”. *IEEE Trans. on Software Engineering*, vol. 20, no.12, Dec. 1994, pp. 948-976.

[22] Parnas, D.L., Lawton, A., “Precisely Annotated Hierarchical Pictures of Programs”. CRL Report 359, TRIO, Communication Research Laboratory, McMaster University, March 1998.

[23] Parnas, D.L., Asmis, G.J.K., Madey, J. “Assessment of Safety-Critical Software in Nuclear Power Plants”. *Nuclear Safety*, vol. 32, No. 2, April-June 1991, pp. 189-198.

[24] Parnas, D.L. ”Inspection of Safety Critical Software using Function Tables”, *Proceedings of IFIP World Congress 1994*, Volume III, August 1994, pp. 270 - 277.

[25] Parnas, D.L., Madey, J., “Functional Documentation for Computer Systems Engineering”. In *Science of Computer Programming,* (Elsevier) vol. 25, No1, October 1995, pp 41-61

[26] Schertzer, W.M., Murthy, C.R. “Physical Limnology and Water Quality Modelling of North American Great Lakes”, Part 2, Water quality modelling. *Water Pollution Research Journal of Canada, V.29, No. 2/3, 1994, Pp.157-184.*

[27] *Wasse*rman, H. and Blum, M. “Software Reliability via Run-Time Result-Checking”. *Journal of the ACM*, Vol. 44, No. 6, November 1997, pp. 826-849.

[28] Qiao, S. “Fast Adaptive RLS Algorithms: a Generalized Inverse Approach and Analysis”. *IEEE Transactions on Signal Processing*, Vol. 39, No. 6 (June 1991), pp. 1455-1459.

[29] Qiao, S. “Error Propagation of Some Fast RLS Algorithms”, in *Advanced Signal Processing Algorithms*, Architectures, and Implementations IV, Proc. SPIE 2027, San Diego, CA, July 1993, pp. 448-454.

[30] Zilitinkevich, S.S., Kreyman, K.D., Terzhevik, A. Yu., “The Thermal Bar”, *J. Fluid. Mech*., vol.236, 1992, pp.27-42.