RAPID DETECTION AND REPORTING OF PROGRAMMING ERRORS IN MEDICAL INFORMATION SYSTEMS

Robert J. Oberst and Janet A. Johnson
St. Louis University School of Medicine
1402 S. Grand Blvd., St. Louis, MO 63104

ABSTRACT

Effective Medical Information Systems (MIS) are characterized by an ability to consistently provide users with information that agrees with their expectations. When a system does not perform as expected it is assumed that this system contains at least one, generally more than one, error. A significant improvement in the Software Development Staff's (SDS) ability to quickly identify errors in a MIS will certainly shorten the development cycle. Errors may be roughly divided into three types: specification errors, logical design errors, and programming language errors. We present procedures that are effective in the detection of many kinds of programming language errors. An example of an implementation of these procedures is demonstrated for a MUMPS multiuser system (Tandem NonStop System).

A BRIEF TAXONOMY OF ERRORS

During the system development cycle the system designer must be aware of the potential occurrence of three types of errors: specification errors, logical design errors, and programming errors. We can readily associate each type of error with a distinct phase of the system development cycle, suggesting that the deficient performance of tasks associated with a particular development phase results in the generation of a characteristic error type. These three phases are of course the specification phase, the logical design/coding phase, and the testing/debugging phase.

In the initial stages of MIS development specification errors may occur as a result of the SDS's misunderstanding the exact needs of end users and the users' inability to articulate those needs in the detail needed for automation. Diligent user involvement in the specification phase remains an effective deterrent to the occurrence of numerous specification errors.

The goal of the specification phase is provision of a complete functional description of a MIS from the point of view of user goals and needs. The succeeding phase of logical design/coding seeks to transform this "conceptual" MIS into an actual working system within the constraints imposed by the choice of a particular computer system and programming language. In recent years a great deal of attention has been devoted to the so-called "structured programming techniques" that are designed to eliminate errors in this phase of MIS development (1,2). It would be somewhat of an exaggeration to claim that adherence to these techniques has revolutionized the logical design/coding phase of MIS development. Nevertheless there is little doubt that such influences have encouraged awareness of and adherence to design and programming standards, leading to increasingly reliable methods of error detection and correction when properly applied.

A WORKING DEFINITION OF PROGRAMMING ERRORS

Since we are mainly concerned with the detection of programming errors in a MIS it will be helpful to present a working definition of programming errors: Whenever the actions of a MIS user result in the unexpected termination of processing, a programming error has been encountered. There are also programming errors which do not cause termination of the program but instead result in such things as endless loops and storage of invalid data. We will not be concerned with that class of programming errors here but will focus our attention on the "fatal" errors which terminate processing. The most reliable method for detecting programming errors involves putting the MIS into operation. This is not to say that there are no other reasonably efficient methods for detecting programming errors. Programming language compilation reveals errors in syntax, the majority of possible programming errors. For languages that are more commonly interpreted such as BASIC and MUMPS there usually exist a number of syntax-checking programs or "preprocessors" that perform a similar error checking function (3). However, even if we assume that pre-processing can in fact detect language syntax errors, we are still left with the problem of detecting run-time programming errors (examples include arithmetic under- and overflow, divide by zero, subscript
Careful testing of the software by those responsible for design and implementation will result in the reduction of many of these types of errors. Further testing by other members of the SDS not involved with the project will reveal unanticipated errors such as those encountered when invalid or unexpected entry formats are used. However, no amount of testing or revision will yield an error-free program of any size or significance. With this in mind we direct our attention to detection and correction of errors in systems installed for production use.

Let us examine traditional approaches to the detection of programming errors for a MIS that is currently in the production phase. Anyone responsible for supporting systems is only too familiar with the following scenario: A call is received from a user of a recently installed MIS who informs the support staff that the system is not working as expected. The following information is carefully elicited from the user:

1. The system in question "bombed" after being presented with a seemingly "normal" set of information.

2. This user originally suspected that he made some type of error in using the system. To confirm this hypothesis the original actions that precipitated the program or problem were repeated an unknown number of times, all with identical results.

3. The user wasted more than 2 hours in futile attempts to coax the system to perform as expected.

4. The last user interaction occurred more than 3 hours prior to this telephone conversation.

5. The only other information recalled by the user is the display of the message "abnormal termination".

6. The user still entertains the notion that he was in some unexplained fashion at least partly to blame for the system malfunction and is reluctant to use it lest he "break" it again.

**OBSERVATIONS**

Experiences of this type combined with previous discussions of programming errors suggest several interesting observations:

-- An understanding of user psychology requires that the SDS inform users of the potential existence of programming errors in a MIS. When a programming error is encountered, the user must be informed that a problem exists and that corrective action will be initiated.

-- The prompt correction of detected programming errors is essential both in order to avoid wasting users' time and to avoid compounding existing errors.

-- It is detrimental to both the SDS and the user for the SDS to solicit participation from the user in the detection of programming errors. Detection and correction of programming errors remains the sole responsibility of the SDS. Programming language considerations must always remain hidden from the user for the following reasons:

1. User perceptions of the causes of programming errors are generally unreliable. Quite often they are based upon mere coincidence instead of an analysis of cause and effect relationships.

2. Users might be expected to acquire information that bears no relationship to their primary areas of expertise.

**TANDEM MUMPS IMPLEMENTATION**

For the past several years the authors have been engaged in the design and implementation of a variety of MIS applications in the areas of Surgery, Internal Medicine and Pathology, as well as Personnel and Administration. All of these applications were developed in the MUMPS language on a Tandem NonStop Computer System. While Tandem MUMPS has proven to be an excellent system development environment for many reasons (4), we will limit our discussion to the area of programming error detection. We have used our insights concerning the ubiquity and persistence of programming errors to develop a flexible programming error detection and reporting system that is easily grafted upon any MIS under development or in production. Activation of the system occurs when a programming error is detected, initiating the following actions:

1. Processing is halted in an orderly fashion and all files currently in use are closed. The following message is displayed to the user:

   *** Processing has been halted because of a programming error. Please do not restart the program before notifying Computer Services. We apologize for any inconvenience that you may be experiencing, and hope to correct this problem as soon as possible.***
2. An option is available to lock the user out of the entire MIS or the part of the system where the error was encountered, until the error has been corrected. While this may seem to many a drastic measure, we have observed a tendency on the part of users to retry an abortive operation a number of times in the hope that the problem will magically "go away". In the event that these unsuccessful operations are responsible for modifying critical data files, the damage done by repetition can be substantial. Operations that merely read various data files can of course be retried without penalty.

3. The following specifics concerning the detected error are recorded: user, time, date, terminal, error type, routine name and actual line in which error occurred, routine stack status and local symbol table contents (optional).

4. The Electronic Mail system is used to send a message to members of the SDS who are responsible for debugging that particular MIS. This message contains the information that was recorded above.

5. For those MIS applications that have been designated as critical, there is the option available to send a message describing the encountered error to the system console, and/or to other output devices to further alert those responsible for program maintenance.

While this programming error detection system has proven to be a useful MIS development tool in a Tandem MUMPS environment, we propose that comparable versions may be implemented for various other multiuser machine and/or programming language environments. However, in order to do so certain requirements must be fulfilled. First, the run-time software must support error trapping. Second, for each MIS application designated SDS must be assigned specific responsibilities for debugging programming errors as they are detected. Third, an Electronic Mail system must be available which is capable of receiving messages from the MIS run-time software, in order that the appropriate SDS may be apprised of detected programming errors.

DISCUSSION

In our investigations of methods for the improvement of MIS design we have encountered an intriguing paradox. Although a vast literature exists describing numerous varieties of MIS that perform interesting and useful functions, little is written concerning the design cycle iterations required before a given MIS performs as expected. We are assuming of course that there are common types of errors of which a majority of programmers are guilty, and that accurate knowledge concerning the characteristics of these errors will eventually promote more effective MIS design. It might justifiably be inferred that we are advocating a more "pathological" approach to the study of MIS design.

There are two important goals we hope to achieve through utilization of a programming error detection and reporting system. First, we wish to eliminate as quickly as possible programming errors that prevent a MIS from performing as expected, with minimal inconvenience to both end users and the SDS. Second, we seek answers to various questions concerning the nature of programming errors. For instance, what types of errors most commonly occur? Are there important differences among programmers in both the types and numbers of errors produced? Is there a significant correlation between years of programming experience and the number and types of errors generated? Although we have only recently begun to investigate these important questions, we hope to report preliminary results within several months. Perhaps this information might serve to improve the skills of future MIS designers.

ACKNOWLEDGEMENTS

The authors wish to thank George E. Gantner, M.D. and Stanley A. Garfield for helpful discussions regarding the ideas presented in this paper.

REFERENCES