I. INTRODUCTION

Computer literature has neglected the topic of program debugging and system testing. Perhaps this is due to programmers' and academicians' adversion to admitting their errors. As a result their experiences in error avoidance are not shared by other users. Yet it is useful and necessary to explicitly and seriously consider the problems involved in implementing programs, procedures and systems.

Jones and McLean (1) state this point quite clearly: "As a class programmers tend to be eternally optimistic. They fail to be realistic about the problems of system design and the time necessary to complete a complex system. For instance, sufficient time is never allotted to debugging; if an appreciable amount of debugging time were to be scheduled at the outset of a project, this would be tantamount to an admission of professional inadequacy." Later in the same paper they suggest that in large scale software system development approximately 30% of the time necessary for system development should be allotted to system integration, debugging and testing. Nearly one third of system development efforts are dedicated to these tasks. In spite of these efforts, literature on the matter is practically unavailable and has been given little attention as an academic subject. Vander Noot (2) points out that: "Out of all the millions of words written over the past few years about EDP, only one article (3) could be found that was devoted to testing, and that was primarily on program testing."

The entire problem of computer implementation from its early design and coding to its actual usage is quite a complex one. This paper is concerned with the steps that take a program which is already designed and coded to the final stages of its actual utilization. These steps could be broken down into testing design and testing implementation. Testing design, which includes testing design for system and program testing, involves the design and preparation of sample data that the program (or software system) will utilize. This data should represent most of the conditions that the program will encounter. In testing implementation, data will be entered for processing and errors will be encountered in coding as well as in the quantitative results obtained. Once programs work independently with this sample data they should be tested in relation to other programs and sample data.

This paper is primarily concerned with implementation testing. APL is used as an online language to illustrate the general typologies of errors that are presented and to exemplify some programming and debugging practices that can be used to make a program (or system) work.

This concern is not unknown to manufacturers. They realize that one of software's most important features is its debugging and error diagnosing characteristics. They also understand that efficient codes are becoming less important with the

The art of online debugging.

Miklos A. Vasarhelyi and Theodore J. Mock

Accounting and Information Systems Research Program
Graduate School of Management
University of California, Los Angeles

Abstract

System testing and program debugging are a somewhat neglected part of data processing. This paper suggests a general taxonomy of programming errors composed of five general categories: A) Grammar and Syntax Errors, B) Logical Errors, C) Misconceptions, D) Integration Errors and E) System Control Errors. Software testing is divided into two main processes: test design and test implementation. Test implementation is also subdivided into two complementary functions: error detection and error clearance (debugging). Two basic procedures are suggested for error detection, while ten different procedures are added to complete debugging. The final parts of this paper discuss the tradeoffs between online and batch debugging and finally the behavioral factors which are related to programmers' errors are discussed. Conclusions and suggestions for further research follow.

I. INTRODUCTION

Computer literature has neglected the topic of program debugging and system testing. Perhaps this is due to programmers' and academicians' adversion to admitting their errors. As a result their experiences in error avoidance are not shared by other users. Yet it is useful and necessary to explicitly and seriously consider the problems involved in implementing programs, procedures and systems.

Jones and McLean (1) state this point quite clearly: "As a class programmers tend to be eternally optimistic. They fail to be realistic about the problems of system design and the time necessary to complete a complex system. For instance, sufficient time is never allotted to debugging; if an appreciable amount of debugging time were to be scheduled at the outset of a project, this would be tantamount to an admission of professional inadequacy." Later in the same paper they suggest that in large scale software system development approximately 30% of the time necessary for system development should be allotted to system integration, debugging and testing. Nearly one third of system development efforts are dedicated to these tasks. In spite of these efforts, literature on the matter is practically unavailable and has been given little attention as an academic subject. Vander Noot (2) points out that: "Out of all the millions of words written over the past few years about EDP, only one article (3) could be found that was devoted to testing, and that was primarily on program testing."

The entire problem of computer implementation from its early design and coding to its actual usage is quite a complex one. This paper is concerned with the steps that take a program which is already designed and coded to the final stages of its actual utilization. These steps could be broken down into testing design and testing implementation. Testing design, which includes testing design for system and program testing, involves the design and preparation of sample data that the program (or software system) will utilize. This data should represent most of the conditions that the program will encounter. In testing implementation, data will be entered for processing and errors will be encountered in coding as well as in the quantitative results obtained. Once programs work independently with this sample data they should be tested in relation to other programs and sample data.

This paper is primarily concerned with implementation testing. APL is used as an online language to illustrate the general typologies of errors that are presented and to exemplify some programming and debugging practices that can be used to make a program (or system) work.

This concern is not unknown to manufacturers. They realize that one of software's most important features is its debugging and error diagnosing characteristics. They also understand that efficient codes are becoming less important with the

advent of larger and faster hardware, and that these codes are becoming progressively prohibitive in the expensive process of program testing and system testing. WATFOR (4), a fast Fortran compiler, was initially developed utilizing the same philosophy as that established by the University of Michigan's MAD language. With MAD the need of a fast compiler for simple educational needs surpassed the necessity of efficient coding for production runs. However, WATFOR as pointed out by Siegel (4) "has practical uses out of the classroom. Chief among these is the time it can save 360 users in that non-productive but, alas, necessary debugging procedure."

The next section discusses a general typology of programming errors. Then, Section III presents routines and procedures which may be used to avoid, diagnose and clear errors. The last sections delve deeper into the issues surrounding debugging including trade-offs between offline (batch) and online debugging, examples of debugging procedures, reporting and diagnosing features, and various behavioral and human information processing factors that induce programmers to err. Finally, the need for and possible routes of future research in the field are suggested.

II. GENERAL TAXONOMY OF ERRORS

Five general categories of errors can be identified. Three of these are related to programming and coding while the fourth is integral to general system design and integration, and the fifth deals with system utilization.

Type A: Grammar and Syntax Errors

Computer languages are still not very flexible. They are not capable of identifying misused or incomplete expressions and operands. Errors related to misspelled variables are commonly found in Fortran, PL/1 and APL. For example, the computer may encounter a variable spelled as DUMY which had earlier been defined as DUMMY. Being unable to identify the latest spelling, the computer would either define a new variable or give a "value error" type message.

Type B: Logical Errors

In this case the logical flow of the program is not well defined or designed, or exceptions and extreme cases are not thoroughly considered and particular cases are ignored, resulting in such absurdities as divisions by zero and square roots of negative numbers. Often the logical flow of a program is incorrect due to improper utilization of the language or simply due to lack of attention by the programmer. Loops that are not closed, transfers to non-existant labels, poor utilization of conditional branching are also errors that occur quite frequently in the programming world.