Massachusetts Institute of Technology
Information Processing Center
Cambridge, Massachusetts 02139

To: Jerry Grochow

RCD

From: John McManus, Bob Daley, Dick Snowberg

Subject: Multics Software Reliability Committee Report

Date: July 20, 1970

CC: Multics Administrative Distribution
(Appendix B)

INTRODUCTION

This report of the Multics Software Reliability Committee outlines how we will measure software reliability, suggests that this same mechanism can be used to measure overall system reliability and discusses improvement of software in general. The ideas presented here and the actions which must accompany them are not intended for only the next eight weeks but rather are a guide for the continuing development of the Multics service.

Multics Software Reliability Index

Although the goals agreed upon in the July 2 meeting are realistic, with the possible exception of 15 minutes mean time to recover, the committee feels that a more comprehensive measure of software reliability is desired. (Perhaps quality would be a better description than reliability; reliable bugs are not desirable.) Also required is a measure which can be expressed as one number and therefore can be plotted versus time to easily show our progress or lack of it. Our measure will be the Multics Software Reliability Index (MSRI). It is computed from five types of errors:

I. Errors which crash the system.

II. Outage of any of the following modules (or the machinery which enable these to operate): pl1, fortran, edm, rename, delete, list, archive, probe, login/logout.

III. Errors* in standard service system commands when no satisfactory alternative exists.

IV. Errors* in standard service system commands when a satisfactory alternative does exist.

V. All errors in non-supported commands including author maintained commands, the scientific subroutine package, machine tools, etc.

VI. Suggestions, complaints, etc.

*Errors are defined as deviations from the Multics Programming Manual.

Type I.	Eight points per hour of downtime the first time a software problem crashes the system, sixteen points per hour the second time a crash occurs because of the same bug and thirty-two points per hour thereafter for that bug.
Type II.	Thirty-two points per day for each outage.
Type III.	Four points per day for each bug.
Type IV.	Two points " " " " "
Type V.	One point " " " " "
Type VI.	No points

Multics System Reliability Index

If there are no objections we would like to expand the scope of this measure of reliability to include the entire system. MSRI is redefined for the "system" and it consists of three components: hardware, software and other. It is a measure of both unscheduled downtime and useability when the system is up. There are several reasons:

1. It is easy to do. The majority of the other (hardware, environment, operator, FE, etc.) problems, as seen by the user, are type 1. That is, they cause the system to be totally unavailable.

2. The hardware working committee is measuring compnent reliability which is to a large degree an internal indicator. Ours is external; both are important.

3. It will insure that hardware/software and undefined failures are accounted for. It will also be easy to change the MSRI component from the hardware to the software columns and vice versa when additional analysis indicates.

4. The comparison of hardware, software and "other" components, as percent of total MSRI will be useful.

Type 1 hardware "bugs" are uniquely identified by device type and failure type. For example: "OP NOT COMPLETE" on memory G is one bug and parity on memory G another. This definition will be used to escalate the points per hour from 8 to 32.

There are many deficiencies in this scheme. To remove these, however, would require an unjustified amount of bookkeeping. What we hope MSRI will show us is our progress toward making a reliable, and therefore a viable, computing service.

Software Improvement

When considering overall system reliability, the fundamental differences between hardware and software failures and their causes must be considered. Hardware failures are due to either (1) design errors, (2) manufacturing errors or (3) wear. The most common of these, wear, does not exist in software. In an existing system which does not have wear, the maximum reliability is achived when no design or manufacturing modifications are allowed except those which correct known design and manufacturing errors. Translated, this means we will have the highest reliability if we make no modifications to Multics software other than to fix bugs. Clearly this is not what we want. The success of Multics software depends on providing additional functions and improving performance (and therefore capacity) as well as reliability. The working committee will try to optimize these conflicting goals instead of worrying only about reliability. For example, it is probably better to push for some major improvements (see appendix A) this summer than to delay these for more orderly installation during the fall semester. The problem, of course, is reaching the proper balance. It is worthwhile repeating a portion of Multics Checkout Bulletin Number 366 by J. M. Grochow and F. J. Corbato': ".... the following recommendations are made: .... That all Multics project personnel rank "bug fixing" as their highest priority work".

Even within the framework described above there are several things, both short and long range, which can be done to improve software reliability. (There are also additional software features which could improve overall reliability such as better handling of hardware errors, but these will not be considered here.)

A. System Testing. Continued effort is required, as always, in the area of system testing. This is true both at the component level (responsibility belonging primarily to Project MAC) and the systems level (responsibility belonging primarily to IPC). Multics has two classes of systems which require different testing techniques:

B. 1. Hardcore/Softcore. These parts of Multics are characterized by the fact that most major bugs are (or should be) detected during testing, most bugs have serious effects on all users and subtle bugs are very likely. Meaningful system tests which detect the subtle errors are very difficult and cannot be performed on the service machine simultaneously with production. The short range solution is to try harder and learn from out mistakes. All programmers should be encourage (forced) to spend as much effort as possible in pre-instllation testing. For the long-term solution, a system to automatically execise new Multics systems under load is desirable - if not requied. This test system would be similar to the script - driven PDP-8 measurement tool and IBM'S TEST-360, but it would be capable of driving Multics with 30 or more simultaneous and different scripts. IPC has hopes of developing such a test tool in the future. A medium range solution which we can implement by September 15 if we decide to, is to re-write the certifier. We also need to further evaluate and possible implement more of the ideas presented by Dave Vinograd in MCB 453.

2. Commands. The commands should be testable during production but before installation. An automated system to check all commands and/or proposed new commands would also be desirable. It may be possible to use the test system described above to test commands. In this mode of operation, the test system would login to the production system as one user who had access to a "command test library" and would test all or any specified commands in his directory. For the short range IPC plans to work on semi-automatic test procedures for those commands which have the largest usage and in which bugs have the largest inpact on users. This will be done as follows:

   a. FORTRAN - Because of the new compiler being written, most bugs will be published and not fixed. Exceptions to this rule will be by a request from IPC Manager of User Services to GE Manager of Language Development.

   b. PL1 - GE is developing comprehensive compiler tests for DO loops. IPC is developing tests for mathematical types of statements and hopes to work on testing I/O statements next. By September 15th IPC should assume responsibility for PL1 installations.

   c. EDM - This is the second highest priority command for which formal test procedures will be developed. Other commands specified in type 1 bugs will follow.

B. Communications. This problem is magnified by the geographic distance between IPC and MAC and between Multics users and both IPC and MAC. Again there are several areas in which we should devote more attention.

1. System/User Communication. We need to continue to keep users informed of non-transparent changes. This should be done before installation, along with the installation date, and at approximately the time of installation an MPM update should be published. Although we are supposedly doing this now, our procedures need to be improved for more rigid enforcement. We also need to investifate and evaluate Tom Van Vleck's suggestion for a semi-automatic method of recording a description of each command change and time of installation.

2. User-problem/System Programmer/Resolution Communications - This loop needs to be shortened considerably. Suggestions are welcomed.

3. IPC/MAC Communication. Some permanent "working committees" should be formed which will meet on a scheduled basis to force us to communicate. For instance, the Daley-Steinberg-M^{c}Manus-Others committee should meet weekly to discuss programming priorities (Why are you going to do that? What should we be doing? Which is most important? How can we help?) and pending changes (Which way should we screw the users this week?)

Our motto during this period of intensive concentration on reliability: "The bitterness of poor quality lingers much longer than the sweetness of met schedules."

Suggestions and critcisms are welcomed. Please direct them at M^{c}Manus. We will very shortly start computing and publishing the weekly MSRI. (The list of bugs on which this is based is in>udd>sl>jws>softbug.info) The other suggestions will be implemented as our combined wisdom and resources permit.

JJM:ilc

Appendix A - Major Multics Development Activities for Summer 1970

I. Performance improvements

1. Tune pre-paging
2. Re-implement fault handling
3. New file directory control*
4. Error recovery*

II. Functional and capacity improvements

1. TTY/ARDS dim*
2. DSU170 (IBM 2314) dim
3. User search rules
4. Dartmouth system
5. LISP
6. Save/resume or
   Absentee runs or
   New IPC
7. Non-standard tape dim
8. Accounting limit stops

*These should also improve long-range reliability

Appendix B - Multics Administrative Distribution

Information Processing Services

R. H. Scott

J. M. Grochow

Information Processing Center

W. J. Burner

J. J. M^{c}Manus

L. J. Ryan

J. R. Steinberg

T. H. Van Vleck

Project MAC Multics Group

F. J. Corbato'

R. C. Daley

J. H. Saltzer

General Electric Multics Group

C. T. Clingen

J. W. Gintell