To: MTB Distribution

From: Gary C. Dixon

Date: November 24, 1981

Subject: Results of the Multics Software Support Study

INTRODUCTION

This MTB examines several functions which could be performed by the Multics Software Support (MSS) unit to improve customer satisfaction with Multics. The functions lie in the areas of:

- improving site support capabilities, and
- increasing the frequency and/or quantity of bug fixes shipped to sites.

This MTB reviews various options available in each area, and concludes that MSS should take on more C&F responsibility so that more errors will get fixed, and should use a method such as bug fix release tapes to distribute fixes to sites in a more orderly and timely fashion.

Comments on this MTB should be directed to Gary Dixon in one of the following ways:

System M continuum:
>udd>m>GDixon>meetings>Multics_Support
(short name mss)

System M mail: GDixon.Multics

HVN: 341-7295

IMPROVING SITE SUPPORT CAPABILITIES

One way to improve the product is to improve MSS's ability to provide direct support to sites. This includes several options:

1) add new tools to the support tool box, and enhance existing tools. Upgrade some existing tools for installation in the system libraries. Such tools might include: a full-capability dial_out facility, with msg packet transmission protocols for shipping source and object modules; enhanced, more-maintainable version of copy_dump_tape and compare_dump_tape; adding new functionality to library maintenance tools (eg, a tool to compare hierarchies on MIT and System M); etc.

2) provide SiteSAs with better information on how to deal with Site problems, who to interface with in MSS, how to use TR system, etc. This could be as formal as a SiteSA Reference Guide, or as simple as a series of info segments which SiteSAs could dprint.

3) have MSS personnel learn about more areas of the system, so they can better deal with problems in those areas. Areas of interest include: crash analysis; system installation methods; system tape generation; running a test system; online debugging techniques; performance measurement & tuning; system administration and operation techniques; emergency recovery techniques; etc.

It is clear that all of the available person-power (and more) could be expended performing the above functions. While most are not on-going functions, history indicates that another list of similar tasks will have been developed once the above tasks were completed.

The real question is: would the improvements in site support capabilities because of one or more of the above options justify the cost of performing these functions? This point is discussed in the "Recommendations" section below.

BUG FIXES

Another way in which MSS could improve the product is by making more bug fixes available to sites, and to get bug fixes to sites sooner. This includes several options:

4) move the C&F (continuation and fix) project from Bob Mullen's Languages and Command System Unit into MSS. Several people could be assigned to the Phoenix-based C&F project, with the responsibility to fix bugs in dormant software. For purposes of this discussion, dormant software is software which is not under active change or development.

5) provide bug fix releases (BFRs) for shipment to sites. The primary purpose of such releases is to shorten the cycle between the time an error is reported in Multics software and the time the fix reaches the customer site. BFRs would be created at 3-4 month intervals between major system releases to distribute fixes for critical and high priority problems and for other dormant software modules.

Options 4 and 5 are discussed in more detail in the following sections.

FIXING BUGS

Most people recognize that Multics could do a better job of maintaining products after their initial development is complete. By moving the C&F project to MSS, we could apply more resources to fixing errors. At the same time, we would be freeing up some resources at CISL for more development activities.

In general, the support level for dormant software does not involve adding new functionality. However, it may occasionally involve significant code rewrites to correct reported bugs, or to upgrade a module to use new system facilities or to improve performance.

Dormant software modules can be divided into two types: modules whose maintenance is assigned to a specific developer; and modules not assigned to anyone (and therefore the direct responsibility of the C&F project). Fixes for each type of module would be handled differently. For unassigned modules, MSS would take total responsibility for implementing the fixes and auditing the changes.

For modules assigned to a developer, MSS would make fixes under the existing policy for "Fixing Minor Problems in Multics Software", MAB-036. This policy requires the developer to approve of such fixes before they are implemented, and to audit the changes after implementation. This policy insures that the proposed fix is proper for the module as it stands now, and that it will fit well with any future development plans.

Clearly, we could reallocate all available person-power to the C&F project and still not fix all of the bugs in dormant software. Some care must be taken in deciding which bugs get fixed, and how much resources to spend on bug fixes.

BUG FIX RELEASES

Before discussing Bug Fix Releases (BFRs), we must define some terminology.

BFR TERMINOLOGY

When a Multics release is shipped to customers, it becomes the _current release_. Development work from that point is applied to the _next release_ of Multics software.

Earlier, we defined _dormant software_ modules as those not under active development for the next release. Conversely, _active software_ modules are those which are being changed for the next (or some future) release.

CONTENTS OF BFRs

Several considerations arise when determining what kinds of changes should be shipped in Bug Fix Releases.

No New Functionality

The term "new functionality" is somewhat ambiguous. In the context of BFRs, one possible definition of new functionality is:

- enhancements for existing software modules which are significant enough to be listed as PFS items, or
- new commands not distributed in the current release, or
- new subsystem requests not distributed in the current release, or
- support for new language features in a compiler.

It should be noted that the items above are not an exhaustive list. Other kinds of changes might be considered "new functionality" in the context of BFRs. Also, other definitions of new functionality could be used in the BFR context. For example, a more stringent definition describes new functionality as any coding change which requires a change to user documentation.

Regardless of what definition of new functionality is used, BFRs should not include new functionality for several reasons. From an HIS standpoint, HIS policy prohibits shipment of new functionality in bug fix releases. This policy stems from revenue considerations.(1)

From an MSS standpoint, adding new functionality to BFRs makes the BFRs more difficult to checkout prior to release. BFRs are applied to current release software, and LISD does not run a current release system on which such bug fixes could be exposure-tested. Thus, adding new functionality increases the risk of introducing new bugs in place on the ones being fixed.

From a documentation standpoint, there are no plans to have documentation changes (ie, addenda for manuals) associated with BFRs. Thus, any new functionality would have to be

---

(1) Roger von Seeburg suggests that Marketing could have all customers include in their control an agreement to pay new rates whenever the new functionality was distributed in a general release. This possibility might allow us to bypass the HIS policy on shipment of new functionality in BFRs.

upwards-compatible with existing functionality, and would essentially be invisible to the sites. It would be difficult to insure that such upwards-compatibility is maintained.

From the site's standpoint, many sites are leery of functional changes to modules because of the impact such changes can have on their users. Such sites thoroughly test releases containing new functionality, using testing produces which require significant time and resources. These sites would probably not want to spend the resources for such testing every 3-4 months (the planned frequency of BFRs), and therefore would avoid applying BFRs (or would attempt to apply only parts of BFRs). Having sites partially install BFRs, or not install them at all, is not a good idea. It would make tracking of the various versions of a module difficult for site and TAC/LISD personnel during error diagnosis.

The key criterion for determining whether a change represents new functionality is the impact such change would have on users. If a control argument is added to a command but users can continue to use the command without knowledge of the control argument, then that control argument does NOT represent new functionality. Updated documentation describing the control argument will not be shipped with the BFR release, so users will never know that the control argument exists (without looking at the source) until the documentation is updated in the next major release. However, if addition of the new control argument involved rewriting of substantial portions of the program, such recoding increases the likelihood of new bugs in the software and would therefore have a greater impact on users.

Clearly, the determination of what changes are eligible for distribution in BFRs is subjective process. Perhaps a small review board would have to be established to make such determinations.

<u>Types of Bug Fixes</u>

In producing BFRs, we must select the contents of the release from one of several possible content levels. The content level chosen has an impact on the amount of work needed to create and checkout a BFR.

A) BFRs could contain only fixes to critical problems in active and dormant software. If such fixes were made to dormant software modules, the same fix could be applied to the current release (via BFR) and to the next release without additional labor. However, when fixes were needed in active software, extra effort would be required to retrofit the fixes to the current release software. BFRs at this level are <u>critical fix BFRs</u>.

B) BFRs could contain fixes to critical problems (A), plus fixes to high- and normal-priority problems in dormant software modules. BFRs at this level are <u>dormant fix BFRs</u>.

C) BFRs could contain fixes to critical problems (A), plus fixes to high- and normal-priority problems in dormant software (B), plus selected fixes to high- and normal-priority problems in active software. For active software modules, the developer would select which problems to fix, based upon impact of the problem and cost of retrofitting the fix onto current release software. BFRs at this level are <u>active fix BFRs</u>.

Critical Fix BFRs

We currently provide critical fix BFRs (usually in the form of source-line changes) to sites on an individual basis when they encounter a critical problem. What is needed is a uniform method of getting all such fixes to every site. Possible methods are discussed below under "Distribution of BFRs".

Dormant Fix BFRs

If we move the C&F project to MSS in Phoenix (option 4 above), then the number of fixes to dormant software will probably increase. By definition of dormant software, the version of the modules being fixed is the same in both the current and next release. Therefore, MSS-supplied fixes to dormant software could be included in BFRs with little or no extra effort (beyond the C&F effort on next release modules).(1)

In addition to efforts of the C&F project, some dormant software fixes are made by other developers as part of their software maintenance responsibilities. If appropriate care was taken, these fixes could also be included in BFRs. However, often such fixes are made in conjunction with software enhancements (new or changed functionality). Care would be required to insure that new functionality was not included in BFR modules.(2) The

(1) In producing bug fixes in dormant software to be applied to both the current and next release, some care must be taken to avoid using new, next release features to fix the bugs. However, past experience indicates that the use of new, next release features is unlikely in such circumstances.

(2) Avoiding new functionality in BFR modules requires more care than might be expected. Consider the case in which new functionality is installed in a module on System M for the next release. A bug is found in this new functionality, so a

prohibition on new functionality reduces the number of dormant software fixes which other developers are likely to provide.

Active Fix BFRs

Including fixes in BFRs for active software modules becomes more expensive. In some cases, developers might have to deal with up to four versions of their software: the current release version with BFRs applied; the next release version (installed on System M); the EXL version; and a development version to which changes are first applied. A significant overhead is required to mentally switch between such versions. In other cases, however, there might be fewer versions (no EXL version, for instance), and a given module might be the same in the different versions, facilitating bug fixes in all versions.

Multiple software versions can further complicate the task of retrofitting a fix from the development version back to current release software, because is sometimes necessary to use a completely different fix approach in the current release software. Development of such separate fixes for current release software can represent a very significant additional overhead.

In estimating the costs involved in dealing with an additional version of the software, and in retrofitting fixes to current release software, most developers of large, active subsystems (eg, compose, PL/I, answering service) felt such fixes would require 30-50% of their time. This overhead reduces the amount of new development work which can be performed. It is doubtful that MDC can afford such cuts in new development work.

Bug-Fix-only MSCR is submitted to solve the problem. It would be easy to mistakenly include this bug fix in the BFR, even though it is fixing a bug which does not exist in the current release.

DISTRIBUTION OF BFRs

Several methods of distributing bug fix releases are possible.

X) maintaining a list of compare_ascii changes in a System M data base. The data base would include information such as:

- bug fix number
- date fix created
- date fix last updated (in case the fix has a bug)
- description of the bug
- TR numbers associated with bug
- error list entries associated with the bug
- new System Technical Identifier (STI) associated with the bug fix
- numbers of prerequisite bug fixes (fixes which must be applied before this bug fix)
- description of testing performed for this bug fix, so far
- description of the bug, as a compare_ascii of current release module version module containing bug fix.(1)

Sites could pick and chose bug fixes from this data base, depending upon: resources at their site to apply such fixes; importance of product being fixed to the site; impact of the bug on site's users; level of testing of the fix; etc.

Y) an informal "release" consisting of a hardcopy list of the complete database described in (X) above, distributed to each site on a monthly basis.

Z) generation of a BFR tape containing modified source and object archives, bound segments, and an exec_com to apply the modified to the system libraries (and to generate a new Multics System Tape (MST) if necessary). A minimal SRB would describe bugs fixed by the BFR, and outline the procedures for installation. Microfiche listings for the changed modules could also be provided.

(1) My thanks to SiteSA Jim Homan for the details of this idea.

LACK OF EXPOSURE TESTING FOR BFRs

The biggest problem which MSS will encounter in producing and distributing BFRs is the lack of exposure testing of the fixes. Current release software is not run on any production system available to MSS, so no system is available on which exposure testing could be performed.

MSS will probably run general regression tests against each new BFR release to insure that major system functionality is not disrupted by bug fixes. However, specific testing of each bug fix after integration into a BFR system is probably not possible, given our current resources.

One possible method of gaining some exposure would be to have our Beta test sites (eg, Ford) which are running the current release apply the BFR changes before the BFR is released to other sites. This technique could limit the impact of any problems which were found. However, this would also delay distribution of fixes to sites. It is not clear that the benefits of a brief exposure at beta sites would outweigh the costs of such delays.

RECOMMENDATIONS

Given the above considerations, it is my recommendation that resources be allocated toward fixing more bugs (option 4) and getting bug fixes to sites more quickly (option 5). While I'm sure some manpower will be devoted to building site support tools and to training MSS personnel, such tasks should be performed only as needed and as time permits.

With respect to BFRs, I would recommend a combination of options B (critical/dormant fix BFRs) and Z (BFR tapes).

DON'T ALLOW SITES TO PICK BUG FIXES

Option X above is probably the easiest for MDC to implement because it avoids the necessity to create and ship BFR tapes, for elaborate system integration testing, for creation of microfiche, etc. However, option X has several drawbacks.

Option X requires more involvement by the SiteSA in applying bug fixes. He must integrate the bug fix into the unmodified source code by hand using compare_ascii output. The potential for errors in application of fixes is thereby increased.

Even worse, the SiteSA must also update the STI data base (>t>psp_info_) and the STI information in bound segments for each PSP to allow offsite maintenance personnel to track which bug fixes have been applied. Procedures for doing this are not well documented.(1)

Option X makes it possible for SiteSAs to get bug fixes sooner, and allows them to pick which bug fixes to apply to their system. While this is an option X advantage, it is also a disadvantage, for it makes it much more difficult to track which fixes have been applied at a given site. In general, STIs apply only to major groupings of software modules (an entire PSP or the Multics System Software Extensions (>sss and >tools), etc). To properly track a pick-and-choose style of bug fixing, we would have to associate STIs with each module in the system. This does not seem feasible at the current time. Therefore, option X requires that we lose the ability to track bug fixing through STIs (this is the current situation when emergency fixes are shipped to individual sites), and that we require a site to apply all fixes covered under a given STI. Neither of these alternatives seems desirable.

---

(1) According to Frank Martinson, is it against LISD policy for anyone but MDC to change STI numbers.

Option Y shares most of the advantages and disadvantages of Option X. It does provide a uniform method of notifying all sites of the availability of bug fixes. However, it shares the problems of application of fixes by SiteSAs and of loss of bug fix tracking via STIs.

Option Z bypasses most of the drawbacks described above. It removes the capability for a site to pick and choose which fixes to apply.(1) In exchange, it simplifies the application of bug fixes by providing modules to which bug fixes have already been applied, plus simple exec_coms for installing such fixes at each site. Such BFRs could be applied by site personnel without the aid of a SiteSA.

Option Z would still require changes to the MSCR form, such as: a BFR box, reference to TR being fixed and/or error list entries being resolved. The BFR box would be needed to distinguish between bug fixes made to current release software and fixes made to next release software (ie, fixes to bugs not yet in the field). The TR and error list information would be needed to include a list of bugs being fixed in the mini-SRB associated with the BFR.

The disadvantage associated with options X and Y of not being able to track bug fixes with STI make these options undesirable. Option Z does not have this disadvantage, because it packages all bug fixes together as a single installation unit.

DON'T RETROFIT BUG FIXES TO ACTIVE SOFTWARE

Spending development resources to fix critical- and high-priority problems in active software (option C) could be expensive. From July 1, 1980 to the present, 46 critical- and high-priority problems were entered as TRs. Most of these were entered against active subsystems. Therefore, it is reasonable to assume that 40-60 such TRs might be entered during a typical inter-release period. Retrofitting fixes for such TRs to the current release could require 25-50% of development resources currently allocated to these active subsystems. Expending such a large percentage of total resources for bug fixing is not reasonable.

Also, we do not have sufficient personnel to add 1 or 2 people to each development project for the purpose of bug fixing. Of course, developers would still have the option of retrofitting

(1) In theory, a site could still choose to install some bug fix modules, and not others. However, most sites would not go to the lengths of to do such chosing (selective retrieve of only certain modules from the BFR tape, modification of the installation procedures, etc).

important bug fixes and distributing them via the BFR tape, if the need arose. However, standard policy would not require that active software bug fixes be retrofitted to current release software.

DO MOVE C&F INTO MSS

The C&F project should be moved to the MSS unit. MSS has more resources available for fixing bugs in dormant software (option 4, above) than do the developement units. Such a move would increase the number of bugs which actually get fixed, provide educational coding opportunities to Phoenix personnel, free up manpower in the development units for other development work, and would produce a positive PR impact with customers. This should also help to improve our goaled TR response times (many overdue problems occur in dormant software that no one has time to correct).