CMU-CS-85-152

A House of Cards:

A History of the Inorganic Evolution of the CMU Bboard System Software

Nathaniel S. Borenstein
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The Carnegie-Mellon University electronic bulletin board system is a vast patchwork of programs running on multiple machines and operating systems, creating the illusion of a network-wide information database where none actually exists. The system is extremely heavily used, with hundreds of generally satisfied users each day. Yet its history is an example of uncontrolled and undesigned software evolution at its purest. In this paper, the workings and evolution of the bboard system are explained, and perspectives of the programmers maintainers are used to draw conclusions about the process by which user-oriented software is designed.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory Under Contract F33615-84-K-1520.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Table of Contents

1. Introduction

At Carnegie-Mellon University, a large community of many hundreds of technical and non-technical users read and post messages every day on the electronic bulletin board system, known locally as "the bboards." The bboards constitute a general community resource by which nearly all important information regarding events and opinions are shared, and reliance on the system is sufficiently complete that many important events are never publicized in any other media. Yet, beneath the surface of this vital community resource is a set of poorly structured programs for which no programmer is willing to take responsibility, and which no one fully understands.

In this paper, I will present an overview of the CMU Bboard system, its user interface, history, and internal structure. I will discuss possible lessons from its development, both for future electronic message systems and for the future evolution of user-oriented software systems.

In such a paper as this, an author's possible sources of bias should be made explicit. I voluntarily maintained a large portion of the bboard system programs for several years while a graduate student at CMU. The experience was both highly rewarding, due to the gratitude and appreciation with which significant improvements in the system were always greeted, and deeply frustrating, due to the obscure nature of the code and the total lack of structure in the system as a whole.

2. What the System Looks Like

The CMU bboard system is an electronic analogue of a set of physical bulletin boards, as might be found on any college campus. Separate bboards exist for a wide range of topics, including the "general" bboard for notices of general interest, the "music" bboard for information about concerts and other music-related news, an "opinion" bboard for general and windy argumentation about any topic at all, and dozens of others. Users can post messages on the bboards containing any information they deem appropriate, but social pressure usually suffices to minimize the numbers of grossly inappropriate messages. A modicum of security measures have been taken to insure that each message is accompanied by the correct name of the person posting the message. These messages are easily but rarely circumvented.

The system simulates a network-wide database by roughly duplicating it on at least 30 machines, running three major operating systems and several variants of these.[1] The database itself is

[1] The system runs on UNIX, TOPS-10, and TOPS-20, including at least two versions each of UNIX and TOPS-10. Expansion of the system to other machines running VMS and Accent is under current consideration. UNIX is a trademark of AT&T Bell Laboratories. TOPS-10, TOPS-20, and VMS are trademarks of Digital Equipment Corporation.

represented entirely differently on the different systems, and on two of these systems (UNIX and TOPS-10) is even duplicated in multiple forms. Further, at least 13 entirely different programs are available for reading the bboards, with dozens of variant versions of some of these. Some of the systems are supported to a large extent by the Facilities staff, while others are supported entirely by individuals or small user communities. The number of programs that can be used to post messages on the bboards is only slightly smaller.

In the sections that follow, I will briefly describe the differing user interfaces available for TOPS-10, TOPS-20, and UNIX users of the bboard system. Such descriptions will be far from comprehensive, but are intended to convey the variety of flavors of interface available.

2.1. The TOPS-10 User Interface

TOPS-10 users of the bboard system can read messages with either of two programs. The BBOARD program take a set of bboard names as command line arguments, and a set of option specifications from both the command line and an initialization file. With no options or arguments, it simply prints all of the notices on a set of default bulletin boards, not even pausing as it fills up the screen. Options available include "/more", which causes the system to pause after each notice and after each full screen, "/since:<date>", which causes the notices to be printed in forward chronological order, beginning on the specified date, and a host of less important options, some totally frivolous. When the "/more" option is used, several additional commands are available to the user each time the system pauses, including sending electronic mail in response to notices, skipping notices, moving backwards in the bboard, and so on.

The other program available to read bboards on TOPS-10 is the RdMail program, a mail reading system which allows users to read bboard notices as if they were incoming mail, more or less. Headers of all new notices are printed initially, and the user can then request to see certain notices with explicit commands (e.g. "t17" to see the message numbered "17" in the headers list). By providing the bboard reading capability within the mail system, the system easily provides such features as sending mail in response to notices and forwarding notices to other users.

The situation for posting messages on TOPS-10 is similar; a POST program exists, which allows the user to specify the subject and bboard name, and then invokes a text editor to allow the creation of the message. RdMail users can also use a "post" command, which posts messages on a bboard by sending mail to an appropriate address.

2.2. TOPS-20

The situation on TOPS-20 is considerably more muddled, for no good reason. Bboards on TOPS-20 are implemented simply as mail files, so that any mail-reading program may be used to read them. However, special programs with features designed particularly for bboard use have also been implemented. Nearly all of these systems present a very similar interface, largely resembling the TOPS-10 bboard program. However, they each include a few special features, and are maintained independently. Many TOPS-20 bboard users are unaware even of which system they are using, as there are three programs named BBOARD, and the one being run depends entirely on the individual's search path for programs to run. In addition to the three separate but similar BBOARD programs, the mail reading programs MM and MS are used for reading bboards, as is the Emacs mail reading package, BABYL. Posting on bboards is done by sending mail, usually with a "post" command within the bboard-reading programs.

2.3. UNIX

On UNIX, a simple system utility program called "bb" can be used to see an entire bboard in reverse-order. When users get past the new notices, they can abort with ↑C or flush output with ↑O. Alternately, they can pipe the output of bb to the "more" utility, to see it a page at a time. Some users pipe the output to the stream editor sed, using a sed program ("bb.sed") to divide the bb program's output into single-notice chunks, marked in such a way that the "more" program can read them one notice at a time. Still other users use a program called "bboard", which simulates very closely the TOPS-10 bboard program's user interface.

However, all of these users are in a minority in that they run a bboard program outside of the Emacs editor. Most UNIX users at CMU do all of their work inside Emacs, and a multitude of different packages for reading bboards within Emacs exist. An old version, known as "10bb", simulated the TOPS-10 BBOARD program within Emacs. Although that program no longer exists in the standard Emacs library, several users have their own private, modified versions which they still use. Another version, known as "bbemacs", divides the user's screen into two windows. One window is filled with bboard headers, one per line, which the user can scroll through using standard editor commands. Pressing space on a given line causes the corresponding notice to appear in the other window. This system provides a large number of supplementary functions, each invoked by a single keystroke in the headers window. bbemacs makes use of a hidden program, ebb, to access the underlying bboard database. Finally, the emacs "bags" program presents an interface much like that of bbemacs, but with still more features, and with the capacity to use the same interface for mail, bboards, unix netnews messages, and cboards. (Cboards are another recent addition to the family of message

systems at CMU. Basically, they are bboards structured by the date of an event rather than the date on which a notice is posted.) Each of the Emacs systems exists in many private, customized versions.

3. How the System Works

Despite the lack of a coherent design, the fact that the database is imprecisely duplicated in multiple and incompatible formats on at least 30 machines, and the fact that dozens of different programs access those databases in an uncoordinated manner, the bboard system seems to function reasonably well. In this section, I will present an overview of how the system is implemented.

3.1. Inter-Machine Communication

Since the database is structured entirely differently on the different operating systems, the only point at which any overall structure can be seen is in the inter-machine communication. Notices are posted on a bboard by sending mail to a unique address on each machine with a bboard system. On UNIX and TOPS-10, all the mail is received at a single address, called "unique-bb". Among the header lines included in such mail is an "Attention:" header, which specifies the bboard on which the notice is to be posted. Thus, a header "Attention: general" will cause the mail to be posted on the general bboard, which is also the default.

Unfortunately, the TOPS-10 and UNIX implementors had slightly different ideas about the function of the "Attention:" line. On UNIX, the assumption was that a notice to be posted on several bboards would only have to be mailed once, with two Attention headers, while on TOPS-10 it was assumed that each mailing would be a single notice for a single bboard. This led to situations where a system crash on TOPS-10, handled conservatively, often caused multiple identical "Attention:" lines to appear on the same notice. The UNIX system handles such a situation by duplicating the notice on its bboards, so that UNIX bboard readers have often seen notices which came via a TOPS-10 system duplicated on their own version of the bboard. (In the most extreme example, a bug had frozen the "Franzlisp" bboard for over a month on TOPS-10. Each day, "Attention:" lines were added to the new notices, but they were never successfully forwarded to the UNIX systems. When the bug was finally fixed and the messages got sent, each was duplicated 35 times on each UNIX system.)

The bboard software on TOPS-20, and on a few random other sites which selectively receive a few CMU bboards, does not understand the "Attention:" convention, and hence requires that mail for each bboard go to a separate address.

Thus, the top-level design of the bboard system, if it can be called a design, is simply to send each bboard notice as an individual piece of mail to each of the other machines. Since the mail system is

asynchronous, and since some of the machines are down at any given moment, this means that the order in which messages appear on a given bboard is entirely different from one machine to the next, although roughly similar. It also means that the integrity of the bboard system is entirely dependent upon the integrity of mail system. On those occasions when the mail system fails, it is usually easier to track lost personal mail than lost bboard mail; most likely, certain machines have missed certain notices on several such occasions.

Beyond the top-level protocol of mailing bodies and including "Attention:" lines, which isn't even itself universally adhered to, there is no common structure to the three different operating systems' implementation of the bboard system. Each will be described separately in the sections which follow.

3.2. The TOPS-10 Implementation

For each bboard on TOPS-10, there are three important files. Because of the limitations on TOPS-10 file names, these are indexed by the first six letters of the bboard name. (This limitation has, on occasion, required changes in the names of bboards which were started first on UNIX. Each bboard name in the overall bboard system must be uniquely identifiable by its first six letters if it is to communicate successfully with TOPS-10.) For the general bboard, for example, the file "GENERA.BBD" contains the text of all of the notices in the bboard, in chronological order. The file "GENERA.DIR" is an index file containing byte counts and pointers into the larger file, allowing the BBOARD program to access notices without reading the entire file. The third file, "GENERA.MSG", duplicates all of the notices, this time in the proper format for reading by the RdMail program.

When a TOPS-10 user posts a message on a bboard, the POST program itself updates the .BBD and .DIR files, and sends an appropriate piece of mail to all the other machines. Thus the notice appears immediately on the machine from which they are posted. When a TOPS-10 system receives mail for unique-bb (that is, when someone on another system has posted a notice), it is appended by the standard mail mechanisms to a mailbox file. Then, when a batch job which runs every fifteen minutes next wakes up, it runs a program called "REMOTE" which appends the new messages to the appropriate bboard. The "REMOTE" program is itself largely a copy of the POST program; at some point in its evolution, the POST program was copied under the name REMOTE and began a separate history of its own, so that now many changes need to be made identically to both programs.

Meanwhile, none of the mechanisms described above do anything to the .MSG file, the bboard file used by RdMail. This is done instead by a nightly batch job, which reads the .BBD files using a special program, "BBDRDM", which lives in another account and for which the source files are unavailable. The new messages are put into a RdMail-format incoming mail file, and RdMail is then run to read

them into the appropriate .MSG files.

Also, one TOPS-10 machine, CMU-CS-A, is the point at which a host of externally-generated material enters the bboard system. Sites connected to CMU via the ARPA Internet or connected networks can send mail to various addresses on CMU-CS-A, and one of several batch jobs will process the mail and forward it to the appropriate machines (and with the right "Attention:" line) to appear on the appropriate bboards at CMU. Still more batch jobs copy bboards from Stanford and MIT, as well as other sources, to appear only on CMU-CS-A. Finally, a monthly batch job runs which purges old notices from all the bboards and archives them permanently on magnetic tape.

This proliferation of batch jobs, data files, and the like makes maintenance difficult, to say the least. For starters, it means that creating a new bboard is an unjustifiably painful endeavor. Bboards have been incorrectly or incompletely created so often in the past that now a file exists which details all the steps necessary to create a bboard on a TOPS-10 system. That file is reproduced in Figure 3-1 for further insight into the complexity of the TOPS-10 system.

Figure 3-1: How to Add a Bboard on TOPS-10

HOW TO ADD A NEW BBOARD (Assuming that you want the bboard to be called the Foobarbaz bboard.)

1) Contact the authorities on the C and VAXes to have the bboard created on those machines, if it is a system-wide bboard.

2) Create the file Foobar.msg -- you can usually do this by copying the file DUM.MSG, which is left around for that purpose. Make sure not to create the file on TEMP:, and to give it protection <155>. You must do this on both the A and the B.

3) Create the empty file FOOBAR.BBD and give it protection <233>. Do not create it on TEMP:. You must do this on both the A and the B.

4) Create the empty file FOOBAR.DIR and give it protection <222>. Do not create it on TEMP:. You must do this on both the A and the B.

5) If the bboard is a CMU-only bboard, then:

5-A) Edit the file BBDRDM.CTL and insert the name SSL:FOOBAR.BBD[F100BB00] in the list of inputs to the bbdrdm program. Then insert a set of lines of RdMail commands to deal with the FOOBAR bboard -- these can be copied from the load, overwrite, etc. commands that exist for each of the bboards. Finally, make sure that FOOBAR.BOX is deleted at the end of the batch run, like all the other BOX files.

If the bboard is a redistribution of an ARPAnet mailing list, then :

5-B) Instead of editing BBDRDM.CTL you need to edit NETBBD.CTL, building a whole page to deal with the new bboard the way the old ones are dealt with. Pay special attention to remailing the bboard to CMUC if you want it to go there.

6) Edit the file BBOARD.NAM[F100BB00]. Use another bboard as an example, and be "SURE" to read the explanation at the beginning of the file. CMUA only will do.

7) Edit the file BBOARD.NAM[A800RD00] and enter a line corresponding to the FOOBAR bboard. This will parallel the other lines in the file in a fairly straightforward way. CMUA only will do here too.

3.3. The TOPS-20 Implementation

The TOPS-20 implementation of the bboard system is remarkably straightforward, when compared to the TOPS-10 or UNIX implementations. This is because the TOPS-20 mailers are "in-place" mail systems, in the sense that incoming mail is appended to mail files when it arrives, instead of stored in an intermediate location. This means that a single file always contains all of the necessary information, and no other file needs to exist. Thus, the bboards exist simply as mail files ("MAIL.TXT") in a number of separate directories ("BBOARD.GENERAL, BBOARD.OPINION, etc."). The cost of this simplicity, however, is that there is no single address to which bboard notices can be sent and sorted by their "Attention:" lines.

3.4. The UNIX Implementation

The UNIX implementation resembles the TOPS-10 implementation in more ways than anyone involved with the system is happy to admit. The basic representation is similar: a ".bb" file contains the raw bboard notices, as in "general.bb", and a ".key" file contains byte counts and pointers by which to pick out individual notices. In another directory, a "header" file reproduces the headers of each bboard notice for use by several of the emacs packages. These files go by names like "h:general", sacrificing the consistency of meaningful suffixes (e.g. ".hdr") for the convenience of having the file names end with the bboard names, which facilitates certain bboard-name-completion features of the emacs packages. The emacs packages access individual notices from the ".bb" files by having the "ebb" program read them from such files and write them into individual notice files in a temporary storage area. Such short files (one notice each) can then be quickly read in by Emacs itself.

When a message is posted by a user on a UNIX system, mail is sent to all of the other machines. Locally, a program called "postmail" is invoked, which actually appends the notice to the ".bb" file and updates the ".key" file. Incoming mail from other systems uses this same "postmail" program to put the mail into the right bboard. The postmail program also updates the "h:" headers files. At regular intervals, a system purge is performed using the "bbedit" program, which purges notices from the ".bb" files and updates the ".key" and "h:" files accordingly.

The UNIX system, though it resembles the TOPS-10 system in its basic structure, has far fewer complications. Adding a new bboard is relatively simple: a few files must be created, and a new entry must be made in a master listing file, which specifies what other machines receive the file in a rather arcane syntax.

4. History of the Bboard Systems

The origins of the CMU bboard system are somewhat obscure and shrouded in folklore. The version of its history described here is gleaned from conversations with many of the programmers involved, and is probably a reasonable approximation to the truth.

Version 1 of the bboard system was, according to legend, a seven-line patch to the LOGIN program on TOPS-10, written some time in the mid to late 1970's. This code simply printed, on the user's terminal, a notice file which contained the bboard notices in reverse chronological order. The user could type ↑O to flush the output once it got to things he had already seen.

Since that seven line patch, the system has *never* been comprehensively rewritten. New features were gradually added, and soon it became obviously desirable to move the code out of LOGIN and into a program of its own. As the size of the bboard (initially there was only one) grew, the cost of keeping the file in reverse chronological order grew; to add to such a file, the entire file needed to be rewritten. In a major cataclysm, the code was redesigned to keep the bboard in forward order, but without provision to allow older users to read it exactly as they had done before. Howls of outrage from the user community caused the system to be put back in reverse order, as it remained for some time. Then the system was more thoroughly redesigned, with the introduction of the ".DIR" file of pointers into the bboard file. At that point, the order of notices was reversed again, into chronological order, as it has remained.

By mid-1980, however, major changes were needed. Nearly all of the CMU Computer Science Department "lived" on the same machine, CMU-CS-A. This machine was so heavily loaded that it was virtually impossible to do anything *but* read mail and bboards on that machine, but it was not possible to do these things anywhere else. Each of the other machines had its own mail and bboard system, but none of the bboard systems talked to each other. (The mail, at that point, had only recently begun to reliably reach any but the TOPS-10 machines.) Each of the TOPS-10 machines had its own bboard copied wholesale to the others every night, but the fact that everything interesting was on CMUA meant that you either had to read things there or remain a full day behind everyone else.

Meanwhile, on the growing number of UNIX machines, a new bb and post system had been written. The interface was, at that point, extremely crude in comparison with what was available on TOPS-10, but the systems did know how to talk to each other. Thus, network-wide bulletin boards existed on the UNIX systems, but everyone remained crowded onto one TOPS-10 machines because that was where everything interesting was happening.

At that point, naturally, a major revision of the TOPS-10 software was necessary to allow it to share bboards with the other machines. Until that time, all of the important information had been posted on the "local" bboard on CMU-CS-A. Each machine had a "local" bboard, but it was the A's local bboard which everyone, by convention, used. In order to make the transition to network-wide bboards, it was necessary to choose a name for the main bboard. Because "local" seemed to imply "this machine only", a new name, "general", was selected. The community was told that, from now on, the main bboard would be the "general" bboard. While no one raised any objections, some people simply never accepted this fact. Four years later, CMU-CS-A oldtimers were still posting notices of general interest on that machine's local bboard.

A new bboard, called the "BBTEST" bboard, was created for testing the multiple-machine software. Unfortunately, one of the features that required the most debugging was the one that used "Attention:" lines to select the right bboard on which to post incoming mail. The general bboard (on which notices were posted by default when no "Attention:" line was present) from that period was continually scattered with messages saying something like "This time it should REALLY appear on the bbtest bboard!" The obvious alternative of making mail go to some other bboard by default was rejected because of the frequency with which important notices actually did arrive with no "Attention:" line for posting on the general bboard.

Another problem raised by the transition was the issue of mail addresses and the integrity of the bboard database. Until 1980, sending mail to "bboard@CMUA" would cause a notice to appear on that machine's local bboard, which reached the entire CMU community. Now, each machine would have a mailing address for receiving bboard notices, but to reach the entire community mail would have to go to *all* of them. Nonetheless, people all over the ARPAnet would continue to send mail to bboard@CMUA when they wanted to post a notice at CMU. The address "unique-bb" was selected as the mailing address for posting on one machine, and a new address, "CMU-BBOARDS", was created for posting on all the machines. There was great debate over what to do with mail that arrived for bboard@CMUA; at first, it was forwarded to the CMU-BBOARDS address, but after a few messages inquiring or complaining about the bboard system were posted system-wide, the mail began to be forwarded to a human maintainer instead.

Once the transition to network-wide bboards was complete, the UNIX bboard system began to undergo the same kind of uncontrolled evolution that had hitherto been characteristic only of the TOPS-10 system. (The TOPS-10 system, meanwhile, was being abandoned in droves and thus became uncharacteristically stable and reliable.) Because UNIX is fundamentally a much easier system on which to program than TOPS-10, the evolution was more horizontal than vertical. On

TOPS-10, a relatively small community of programmers continually added more features and patches onto a few basic pieces of code. On UNIX, virtually every programmer felt free to make his own private version of the bboard-reading code, modified to suit his tastes. This was perfectly satisfactory until bugs were found in the original code; at such times, it was impossible to correct the bugs in all of the versions that had evolved, and there was always someone using the buggy software. Fortunately, the protection mechanisms kept this from affecting the main data files, but the temporary emacs files, stored in a publicly writable area, were occasionally destroyed by outdated software.

5. Conclusions: Lessons of the CMU Bboard System

The CMU bboard system is known intimately by a great many people. There are two basic perspectives with which people view the system, that of the user and that of the maintainer.

Users of the bboard system seem, as a whole, to be extraordinarily happy with it. Many people literally use the system for hours daily, reading multiple bboards and carrying on extended conversations via bboards and mail. Others avoid the system as they would a dangerous disease, for fear of finding themselves spending so much time with it. Calls for major redesign of the system are virtually unheard of in the user community; the only complaints ever made are calls for increased functionality in a particular interface or complaints about rare breakdowns in the system (lost messages, duplicate messages, infinite loops, core dumps, and the like). In short, the system is a hit with its users.

Among those who maintain the system, however, it is considered little short of a disaster. Its uncontrolled evolution has produced a system of hundreds of pages of code where a dozen or two should suffice. While some of the component systems are relatively well-controlled, at least in terms of version control and control over modifications, other parts of the system, most notably customized Emacs interfaces on UNIX, are beyond the reach of any maintainer. Thus while everyone who has ever been intimately associated with the system yearns to see it completely reimplemented, everyone knows the howls of outrage with which such an event would inevitably be greeted, no matter how well done. Meanwhile, the code continues to grow. The author of this paper maintained the TOPS-10 system for about two years, ending in late 1982. I still have no idea how a large chunk of that program works, nor does the previous maintainer. A few individuals in the community know more, but are generally reluctant to admit it for fear they'll have to help fix something.

Such an environment has a snowballing effect. When people modify a system they do not understand, they tend to do so in very crude ways that are inefficient and inelegant but sure to work. For example, the system I found when I became involved with the TOPS-10 program ran a regular

batch job to process incoming mail to various addresses and post it on the appropriate bboards. The incoming mail was first copied to a temporary file for protection purposes. It was then edited with the TECO text editor to remove certain header lines. It was then edited with the SOS text editor to remove certain other header lines, and to very slightly change the way the file was structured. Then, finally, the program REMOTE was run, which read in the file and removed still more header lines. In all, three separate programs were run each time new mail came in. It was only after this batch job was made to run every fifteen minutes that I took it upon myself to figure out why it was so slow. As it turned out, about a half hour's work with the REMOTE program sufficed to eliminate the SOS and TECO runs entirely. But the people who had added the "feature" by which the unnecessary header lines were eliminated had been unwilling or unable to modify the REMOTE program (written in SAIL), and had opted instead for the safe, easy course of running an extra program.

The question that I asked my colleagues who have worked with the bboard systems when I began contemplating this paper was a simple one: "What went wrong? Why is the bboard system such a mess?" A better question, perhaps, is this one: "Did anything go wrong? Can a system so popular with its users possibly be called a failure?" Calling the CMU bboard system a triumph of software engineering would make anyone involved laugh heartily. Calling it a failure will stir many to outrage, so happy are they with its daily use. Yet calling it a success should make anyone who cares about software maintenance cringe.

The common belief of those intimately involved with CMU's bboards, it seems, is that the system is a horrible mess inside but could not have been any different. It evolved, not only without comprehensive design, but without any clear notion of what such a facility could mean to its users. Certainly no one can fault the anonymous author of the seven-line patch to TOPS-10 login for not providing a structure that would be suitable to a network of 30 machines, three operating systems, several major interface styles, and dozens of bboards. He probably just wanted a way of warning people when the system would be going down for maintenance. Moreover, there was never anyone in a position to take the time for a redesign. Facilities staff members could afford to get involved only when the system seriously broke down or created bottlenecks. (Shortly after the bboard system went network-wide, for example, the poor old TOPS-10 mail system, suddenly finding its effective work load tripled or worse, began to develop backlogs of several days, forcing a major effort by the facilities staff to speed it up.) Most of the increased functionality was implemented by graduate students sneaking a few minutes off from study or thesis work; such people were rarely inclined to rewrite anything from scratch. Even today, a comprehensive design would face major conflicting demands and perspectives about how the system should operate.

The larger question is a simple one: which is more important, software that is easy to fix or software that is easy to use? For applications like mission-critical space software or control programs for medical robots, the answer is simple. But for user-oriented applications, the issues are much more complex. It is much easier to build and maintain a simple accounting package than a spreadsheet, but users want spreadsheets. User-oriented software is still in such an early stage of evolution that no one knows what most kinds of systems should look like. Is there, then, any hope to allow such software to evolve with less structural chaos than that which can be seen in the CMU bboard system? There are a few hopeful signs.

The only part of the CMU bboard system which has ever been comprehensively rewritten is the UNIX Emacs interface, which has been rewritten completely at least four times. By the fourth version, the "bags" system, the greatest functionality was provided along with the best structured code -- a natural culmination to a series of rewrites based on increasing knowledge about what the interface should look like. This rewriting process reflects the suitability of Emacs -- which provides very high-level primitives in its extension language, Mlisp -- as a testbed for user interface design. Because so little of the code was devoted to driving terminal displays or managing the underlying database, it was less painful to perform major redesigns and reimplementations of the interface itself. This suggests that providing better high-level tools for interface design may help to reduce the kinds of problems encountered in the evolution of the bboard system. (Emacs itself, however, is a maintainer's nightmare: large mlisp programs tend to trigger horrible bugs, entirely beyond the reach of the mlisp programmer. Nonetheless, a similar tool providing interface-oriented primitives without the bugs holds out enormous potential.)

Another useful tool would be increased communication about the user interfaces of software systems. As the CMU bboard system evolved, similar evolution was taking place at Stanford and in several other computing communities. Undoubtedly the evolution of all of the systems would have benefited by more communication. There is a great need for some kind of comprehensive database about software interfaces, so that, for example, the designer of a fancy mail system for the next generation of microcomputer could look up information about mail system interfaces that have been built before. With a clearer understanding of what has been and can be done, the designer is much more likely to produce a system whose future evolution will not need to be so radical or chaotic as that described in this paper.

Finally, it should be made explicit that a major lesson of the CMU bboard system is that structured programming, top-down design, and all of the other ideals of modern software engineering methodologies, may not be all they're cracked up to be when it comes to the design of user-oriented

systems. Designers of such systems rarely have such deep insight into human nature that they accurately predict what their users will want. It was quite likely the lack of comprehensive control that allowed the bboard system to evolve to a point of such nearly universal popularity. (Indeed, those few components of the system which were more tightly controlled, such as the UNIX "bb" program, were the ones most quickly overrun by alternative programs with more popular interfaces.) Today, the system is in a state of great maturity. It is likely to finally be comprehensively rewritten in the forseeable future, when a network-based central file system will allow a system without the current massive duplications to be created. If such a system is built properly, with an understanding of the special features and multiple interfaces that make the system so popular today, it will probably be possible to design a very clean, efficient, and stable system that is every bit as well-liked as the one now in place. In the world of user interfaces, it is probably best to postpone the software engineering until the true experts -- the users themselves -- have directed the construction of the interface they want. At that point, the software engineers can step in and build it right, instead of merely building the wrong interface in the right way.

Acknowledgements

Several people have helped me with contributions of fact and opinion regarding the history of the bboard system, most notably Craig Everhart, Rudy Nedved, Mark Sapsford, and Eddie Caplan. Their help is gratefully acknowledged.