Usability Testing of a Hyper-M Application

by Aviva Furman and Thomas Munnecke

Abstract

There has been a heightened concern within the M community about the usability of software products. Easily usable software is not a guaranteed byproduct of a graphical user interface (GUI). Several methods contribute to creating software products that are easy to learn and use. One method is usability testing—evaluating the use of a prototype system by representative users performing typical tasks. This article describes a usability test conducted on Hyper-Mail, an e-mail application under development that uses Hyper-M, Science Applications International Corporation's (SAIC) visual-development environment (VDE).

Introduction

Developers of M systems are recognizing the importance of creating sophisticated user interfaces to compete in today's software market. There has been much discussion within the M community about which GUI to use and whether to bind to the X Window protocol or develop an M-windowing application program interface (API). There has been little discussion, however, on just how to go about creating "usable" software. Usability has been defined as the ease with which people in a defined group can learn and use a product.[1] Usable software is not an automatic byproduct of a GUI application, even when that application has been developed using an API. Though APIs do promote a consistent look and feel, developers still have a wide range of possibilities in creating an interface. Usable software evolves from following certain guidelines, principles, and methodologies.

Much has been written regarding principles and guidelines for interface design.[3,4,6] Principles provide suggestions of a general nature such as "Know thy user" or "Engineer for errors." Guidelines provide more specific recommendations, such as "Provide feedback for all user actions." Following principles and guidelines is a good starting point for developers of GUI applications, but even with a thorough knowledge of the principles and a strict adherence to guidelines, designers can go astray. Developers should evaluate the usability of prototype software through actual use by a representative sample of users. Based on the results of the usability test, developers can refine the prototype system (see figure 1).

```
; Usability testing procedure
Design prototype system based on guidelines and
  user input
D  Q:(DEADLINE=+$H)!(PROBLEMS=0)
. Define usability test objectives
. Select representative test subjects
. Design a test scenario which will satisfy
  objectives
. Perform test
. Compile and evaluate feedback
. Incorporate modifications for next iteration
. Q
```

Figure 1. M pseudo-code for usability testing.

Usability Testing and Functional Testing

M developers commonly include functional testing, or alpha-beta testing, as part of the development cycle of a product. On the other hand, developers rarely perform usability testing as a distinct phase of development; this has been important but often omitted. The greater percentage of effort devoted to user interface makes usability testing highly important, however. Usability testing differs from functional testing in several significant ways.

Usability testing focuses on a product's *presentation*, not its *functionality*. The presentation of a product is the way in which it communicates with the user. This includes all of the aspects of the interface: screens, menus, documentation, help text, and error messages.[3] Usability tests most often attempt to pinpoint aspects of the presentation that make the product difficult to learn or use. In contrast, functional testing focuses on the functionality of a product, attempting to uncover bugs and errors in the software.

Ideally, usability testing begins early and continues iteratively throughout the development cycle. Functional testing usually occurs toward the end of the cycle. Early usability

There has been much discussion within the M community about which GUI to use and whether to bind to the X Window protocol . . . There has been little discussion, however, on just how to go about creating "usable" software.

testing actually can contribute to the planning and design of the interface to emulate real-world use. For early testing, the key is to prototype "testable chunks." Iterative testing results in stepwise refinement of an interface. Done in this manner, there should be no major changes necessary as the product shipment date draws near.[2,5]

Usability testing is conducted in a more controlled environment than functional testing. Subjects are chosen to reflect the expected users of the product. Tasks are designed that reflect typical use of the product in a real-world setting. Information regarding the subjects' interactions with the software is methodically recorded by observers or by a video camera. Subjects also may complete questionnaires, which provide additional feedback. Finally, it is important to test enough subjects in order to ascertain whether problems are idiosyncratic or global. In functional testing, developers rarely have an opportunity to observe actual use of the software. The developer must rely upon the testers to accurately report bugs.

The Hyper-Mail Prototype

Hyper-Mail is a GUI e-mail application, developed using SAIC's programming tool for the M environment. The Hyper-Mail prototype is compatible with existing MailMan data structures, and echoes its conceptual structure and functionality. Hyper-Mail has a modern point-and-click windows interface, which makes it more intuitive to use. Hyper-Mail's desktop can be seen in figure 2. Approximately 50 percent of Hyper-Mail's planned functionality was available when the usability study was undertaken. The task list focused on the functions that were available.

Hyper-M is a powerful prototyping tool, supporting "outside-in" development. With Hyper-M, developers create screens and make graphical modifications without writing a single line of M code. These screens are then linked with underlying actions through scripts: M programs that use high-level functions for performing screen manipulation. With Hyper-M, the user interface can be developed independently of M application code. Scripts can be generated quickly using high-level functions for screen manipulation. The net result is rapid prototyping. Additionally, systems can be modified easily based on results of usability testing.

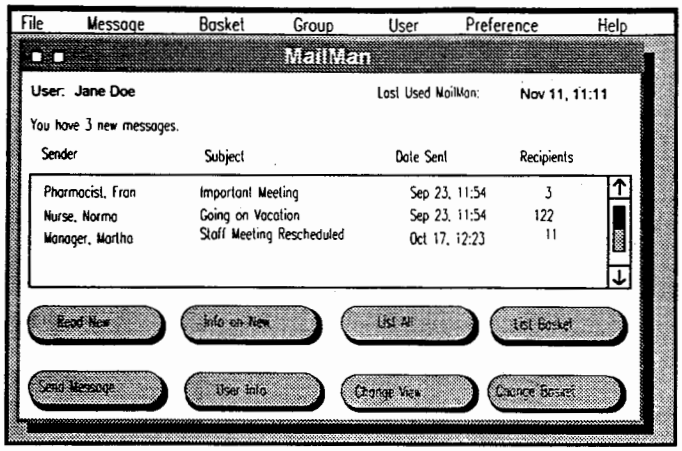

Figure 2. Hyper-Mail desktop.

The Hyper-Mail Usability Study

The purpose of the usability study was to determine aspects of the Hyper-Mail presentation that would interfere with its ease of learning or use. We were interested in determining whether users who are familiar with MailMan's functionality and GUI-style applications could be productive on Hyper-Mail with little or no training.

We chose representative subjects for the usability study, those who would be the expected users of the Hyper-Mail product. We assumed that most Hyper-Mail users would have some familiarity with MailMan. In addition, we chose subjects who had some familiarity with using a mouse and a GUI. These users were best able to evaluate the consistency of the Hyper-Mail interface. In addition, we did not wish to take the time to train subjects on GUI basics given the limited time of the usability test.

Early usability testing actually can contribute to the planning and design of the interface to emulate real-world use.

A total of twelve volunteers participated in the study, conducted in two separate sessions. Subjects in the first session were clinicians and administrators from a local hospital. Subjects in the second session were SAIC employees. We resisted the temptation to recruit in-house programmers. Although readily available and enthusiastic, they did not fully represent the expected end user of the product. Subjects' experience with MailMan ranged from novice to very experienced.

Procedure

The two-hour usability test session consisted of three segments. During the first segment, we introduced subjects to the purpose of usability testing and briefly demonstrated the Hyper-Mail prototype system in what was meant to be a cooperative, nonthreatening atmosphere. Subjects were assured that the purpose of the test was to evaluate the software, not their expertise at using it. We stressed our need to know about any aspect of the interface that they found confusing, noting that if one user was confused, it was likely that others would be. We let subjects know that we valued their suggestions and appreciated their taking the time to participate and contribute to the development of a more usable interface. During a brief demonstration of the Hyper-Mail system, we reviewed basic GUI interaction methods, such as scrolling through a list and selecting a push button.

During the second segment of the two-hour session, subjects individually performed tasks found in the test scenario. The tasks were typical of using MailMan, such as reading new mail, finding a particular message, or looking up information on a user. Each subject was paired with an observer to record the ease with which the subject completed the task, any specific difficulties, and any suggestions offered by the subject. The observers made sure that subjects kept their focus on the task list, helped subjects if they got stuck, and encouraged subjects to suggest improvements to confusing aspects of the interface. Following the tasks, subjects completed a post-task questionnaire designed to capture general feedback about the system and the usability test procedure.

Subjects and observers had a debriefing session about their experience with usability testing of the Hyper-Mail prototype as the final segment. The group discussed reactions to Hyper-Mail as compared with the non-GUI MailMan, ways to improve the Hyper-Mail system, and general comments about the usability test procedure.

Following the test sessions, observers met to discuss and compile their observations. We organized this information into a list of problem areas and possible solutions (see figure 3). In most cases, a problem area was experienced by more than one subject. Sometimes, a problem area was unique to a particular subject and in a few cases, subjects actually had opposing views about an interface issue. Problem areas fell into three sections: system-wide Hyper-M interface issues, systemwide Hyper-Mail interface issues, and specific Hyper-Mail issues. We discussed and recorded possible solutions to each problem area.

Results

According to feedback in the post-test questionnaire and the debriefing session, the overall reaction to the Hyper-Mail interface was positive. Subjects were impressed with its ease of use and felt that users could become productive quickly on the system with little or no training. In comparing Hyper-Mail with the non-GUI MailMan, subjects found Hyper-Mail more intuitive, faster, and more fun. One subject remarked, "When will Hyper-Mail be available? Yesterday, I hope." This sort of encouraging feedback assures developers that they are on the right track.

Though positive feedback is encouraging, the real value of the test comes from isolating problem areas in the interface and obtaining suggestions for improvement. There were several areas that were consistent problems. One systemwide problem involved the ease of accessing different functions. One subject remarked that a user should never be more than one screen away from the desired function. In the prototype system, functions were not always directly accessible. For example, to query the recipients of a message, it was first necessary to read the message. This design flaw was a remnant of the linear conceptual model of the non-GUI MailMan. In a linear model, it is necessary to perform a sequence of steps to invoke an action. In the object-oriented model, a user need only select an object, such as a MailMan message, then select an action, such as querying recipients. The intermediate step of reading the message should be unnecessary. The solution was apparent from watching subjects try to invoke functions. They consistently hunted for the functions on the action bar. By making functions appropriately available on the action bar, users would be able to access them more directly. Figure 3 shows problem areas and possible solutions.

When problems are identified in the development stage, user acceptance is increased and user frustration is decreased.

Other problem areas included occasional lack of appropriate feedback for user actions, instances of unclear or ambiguous text on push buttons, cumbersome methods for selecting multiple items from a list, and inconsistent methods for identifying recipients of a message. In general, feedback from subjects was based on observing what subjects *did* and listening to what subjects *thought*. For example, one might *observe* a subject having difficulty selecting multiple baskets from a list, and then *listen* to a subject's suggestions on how to im-

Issue	Possible/**Implemented** Actions
Subjects had difficulty finding the appropriate push buttons to perform certain options. Subjects repeatedly searched the action bar to find options.	**Make all actions appropriately available from the action bar.** Disable actions when they do not apply to the current state. Make frequently used options available as push buttons.
Subjects wished to receive confirmation after the system performed an action, such as saving messages to a basket.	• Display confirmation message on a reserved area of the screen. No action is required from the user. • **Use pop-up window to confirm that action has occurred. User must press "Enter" to continue.** • Use pop-up window to indicate that action is occurring. Give user the opportunity to cancel before action is complete. • Allow user to select one of the three options through user preference.
Subjects noticed inconsistency between looking up users and looking up groups.	**Enhance group look-up to behave like user look-up.**
Text on certain push buttons was ambiguous or unclear.	• Make push buttons larger so that labels can be more descriptive. • **Rely upon training, online help and familiarity with the new system.**

Figure 3. Problem areas and possible solutions.

prove multiple selection. In fact, many subjects had difficulty selecting multiple items from a list, and they offered a wide variety of suggestions on how to better implement multiple selection. The indisputable lesson is that this aspect of the interface needs improvement; the developers must decide how best to implement the modification. A subsequent usability test could determine if they had made the right choice.

Conclusion

As M systems' developers move from glass-teletype interfaces to GUIs, they should be aware of methods to create interfaces that are easy to learn and use. Although programmers have varying opinions on the best way to achieve these qualities, usability testing is a critical step in the development process. It helps to identify design obstacles and ensure that the end product is indeed usable.

Though usability testing is not without a price, it is certainly cost-effective. When user-interface problems are identified in the development stage, user acceptance is increased and user frustration is decreased. In addition, an intuitive interface requires minimal user training. An ounce of usability testing is worth a pound of user training.

References

1. M. Dieli, "The Usability Process: Working with Iterative Design Principles," *IEEE Transactions of Professional Communications*, 32:4 (December 1989), 272-277.

2. J.D. Gould and C. Lewis, "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM* 28, 3 (March 1985), 300-311.

3. J.N. Mosier and S.L. Smith, "Applications of Guidelines for Designing User Interface Software," *Behaviour and Information Technology* 5(1), 39-46.

4. D. Norman, "Design Principles for Human-Computer Interfaces," *Readings in Human-Computer Interaction, Morgan Kaufmann Publishers*, 1987, 492-501.

5. S. Rosenbaum, "Usability Evaluations Versus Usability Testing: When and Why?," *IEEE Transactions on Professional Communications*, 32:4 (December 1989), 210-216.

6. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, MA: Addison-Wesley Press, 1986. ❖

Thomas Munnecke is assistant vice president of the HyperWare Division of SAIC. He is a past vice chair of the MUMPS Users' Group and is also on *M Computing's* Review Board. He was instrumental in developing Hyper-M, VA FileMan, VA MailMan, and VA Kernel.

Aviva Furman is a principal application developer in SAIC's HyperWare division. She currently is developing Visual FileMan and HyperMail. She has a B.S. from Carnegie-Mellon University and an M.S. from the University of Washington. Her interests include usability, user-interface design, and gardening. She telecommutes from her island home in the Puget Sound. Phone: 206-567-4638. E-mail: afurman@seattleu.edu.