Designing Computer System Messages

Ben Shneiderman
University of Maryland

In our use of interactive computer systems and studies of their users, we have become increasingly aware of the importance of system messages. Novice users are unimpressed with CPU speeds, disk storage capabilities, or elegant file structures. For them, the system appears only in the form of the messages on their screens or printers. So when novices encounter violent messages such as "FATAL ERROR, RUN ABORTED", vague phases like "ILLEGAL CMD", or obscure codes such as "OC7" or "IEH2191", they are understandably shaken, confused, dismayed, and discouraged from continuing. The negative image that computer systems sometimes generate is, we believe, largely due to the difficulties users experience when they make mistakes or are unsure about what to do next.

Several attempts at writing systems to produce more appropriate messages for student programmers [1, 2, 7] and bibliographic file searchers [10] have been made. Golden [5] pleaded for better messages in operating system control languages, and Dwyer [3, 4] offered suggestions on how to improve messages in typical commercial applications. Mosteller [6] reports on error distributions in job control language use.

There are many kinds of messages that should come under closer scrutiny during the design process: menu selection choices, prompts for command language or data entry, feedback indicating completion of a task, results of database searches, and error messages.

To explore the impact of error messages on users, we conducted five controlled experiments [9]. In one experiment, Cobol compiler syntactic error messages were modified and undergraduate novice users asked to repair the Cobol statements. Messages with increased specificity generated 28 percent better repair scores.

Subjects using a text editor with only a question mark for an error message made 10.7 errors, but only 6.1 errors when they switched to an editor offering brief messages. In another experiment, students corrected 4.1 out of 10 erroneous text editor commands using the standard system messages. Using improved messages, the experimental group could correct 7.5 out of the 10 commands.

In a study of the comprehensibility of job control language error messages, students receiving messages from two popular contemporary systems scored 2.9 and 3.8 out of 6, while students receiving improved messages scored 4.8. Subjective preferences also favored the improved messages.

These initial experiments support the contention that improving messages can upgrade performance and result in greater job satisfaction. They have led us to make the following five recommendations for system developers.

(1) *Increase Attention to Message Design:* The wording of messages displayed by a computer system should be more carefully considered. Copy writers or copy editors should be consulted about the choice of words and phrasing to improve both clarity and consistency.

(2) *Establish Quality Control:* Messages should be approved by an appropriate quality control committee. Changes or additions should be monitored and recorded.

Since the error messages that a novice encounters have the most dramatic impact because they appear at a moment of confusion or incomplete knowledge, we have made them the focus of our investigations. In summary, we believe that error messages can be easily and substantially improved.

(3) *Develop Guidelines:* Error messages should meet the criteria outlined in Figure 1. They should

—have a positive tone indicating what must be done, rather than condemning the user for the error. Reduce or eliminate the use of terms such as "ILLEGAL", "INVALID", "ERROR", or "INCORRECT". Instead of "ILLEGAL PASSWORD", try "Your password did not match the stored password. Please try again."

—be specific and address the problem in the user's terms. Avoid the vague "SYNTAX ERROR" or obscure internal codes. Use variable names and concepts known to the user. Instead of "INVALID DATA" in an inventory application, try "Dress sizes range from 5 to 16."

The Pracniques section of Computing Practices presents brief case, method, and problem descriptions of particular interest to the practitioner. See the February 1981 issue of Communications, page 72, for a broader editorial definition.

CR Categories and Subject Descriptors: D.2 [**Software Engineering**]; D.2.5 [**Software Engineering**]: Testing and Debugging–*error handling and recovery*; H.1.2 [**Models and Principles**]: User/Machine Systems–*human factors*.
General Terms: Human Factors
Additional Key Words and Phrases: error messages, system messages, human/computer interaction

Author's present address: B. Shneiderman, Dept. of Computer Science, University of Maryland, College Park, MD 20742.

SYSTEM MESSAGES

SHOULD NOT	SHOULD
BE	BE
—WORDY	—BRIEF
—NEGATIVE IN TONE	—POSITIVE
—CRITICAL OF ERRORS	—CONSTRUCTIVE
—GENERAL	—SPECIFIC
—CRYPTIC	—COMPREHENSIBLE
SUGGEST SYSTEM CONTROL OVER THE USER	EMPHASIZE USER CONTROL OVER THE SYSTEM

OTHER CONSIDERATIONS
- —UPPER- AND LOWERCASE LETTERS ARE PREFERRED TO UPPERCASE ONLY (EXCEPT IN EXTREME SITUATIONS)
- —ASTERISKS SHOULD BE USED ONLY IN SPECIAL CIRCUMSTANCES
- —ERROR NUMBERS, IF NEEDED AT ALL, SHOULD APPEAR AT THE END OF THE MESSAGE
- —USER-MODIFIABLE MESSAGE FILE
- —TWO OR MORE LEVELS OF MESSAGES
- —REDUCE THE USE OF TERMS LIKE "ILLEGAL", "INVALID", "ERROR"

Fig. 1. Summary of System Message Design Guidelines.

—place the user in control of the situation and provide him/her with enough information to take action. Instead, of "INCORRECT COMMAND", try "Permissible commands are: SAVE, LOAD, or EXPLAIN."

—have a neat, consistent, and comprehensible format. Avoid lengthy numeric codes, obscure mnemonics, and cluttered displays.

Writing good messages, like writing poems, essays, or advertisements, requires experience, practice, and a sensitivity to how the reader will react. It is a skill that can be acquired and refined by programmers/designers who are intent on serving the user. However, perfection is impossible and humility is the mark of the true professional.

(4) *Carry Out Acceptance Test:* System messages should be subjected to an acceptance test with an appropriate user community to determine if they are comprehensible [8]. The test could range from a rigorous experiment with realistic situations (for life-critical or high reliability systems) to an informal reading and review by interested users (for personal computing or low-threat applications).

Complex interactive systems, which involve thousands of users, are never really complete until they are obsolete. Under these conditions, the most effective designs are those that facilitate evolutionary refinement. If designers, maintainers, and operators of interactive systems are genuinely interested in building "user-friendly" systems, they must understand users' problems.

(5) *Collect User Performance Data:* Frequency counts should be collected for each error condition on a regular basis. If possible, the user's command should be captured for a more detailed study. If you know where users run into difficulties, you can then revise the message, improve the training, modify the manual, or change the system. The error rate per thousand commands should be used as a metric of system quality and a gauge of how improvements effect performance. An error counting option is useful for internal systems and can be a marketing feature for software products.

Improved messages will be of the greatest benefit to novice users, but regular users and experienced professionals will also profit. As examples of excellence proliferate, obscure, complex, and harsh systems will seem more and more out of place. The crude programming environments of the past will gradually be replaced by systems designed with the user in mind. Resistance to such a transition should not be allowed to impede progress toward the goal of serving the growing user community.

References

1. Conway, R.W., and Wilcox, T.R. Design and implementation of a diagnostic compiler for PL/I. *Comm. ACM 16*, 3 (March 1973), 169–179. Description of the PL/I student-oriented compiler at Cornell University.

2. Cress, P., Dirksen, P., and Graham, J.W. *FORTRAN IV with WATFOR and WATFIV.* Prentice Hall, Englewood Cliffs, N.J., 1970. Textbook for the student-oriented compiler developed at Waterloo University. Includes full set of error messages.

3. Dwyer, B. Programming for users: A bit of psychology. *Comptrs. and People 30*, 1 and 2 (1981), 11–14, 26. Appealing review with many examples of how messages might be improved.

4. Dwyer, B. A user-friendly algorithm. *Comm. ACM 24*, 9 (Sept. 1981), 556–561. Specific suggestions for improving interactive user interfaces using Cobol examples.

5. Golden, D. A plea for friendly software. *Software Engineering Notes 5*, 4 (Oct. 1980), 4–5. A passionate request for improving job control language and system messages.

6. Mosteller, W. Job entry control language errors. *Proc. SHARE 57*, SHARE, Inc., Chicago, Ill., 1981, pp. 149–155. A report on a data collection study which reveals the patterns of user errors for the IBM JES2 system.

7. Moulton, P.G., and Muller, M.E. DITRAN—a compiler emphasizing diagnostics. *Comm. ACM 10*, (Jan. 1967), 45–52. An early report on an attempt to build a student-oriented Fortran compiler.

8. Shneiderman, B. *Software Psychology: Human Factors in Computer and Information Systems.* Little, Brown & Co., Boston, Mass., 1980. Professional book which makes the case for controlled, psychologically oriented experimentation in programming, database use, and interactive systems.

9. Shneiderman, B. System message design: Guidelines and experimental results. In *Directions in Human-Computer Interaction*, A. Badre and B. Shneiderman, Eds., Ablex Publishing Co., Norwood, N.J., 1982. A more in-depth discussion of system message design, presents five brief experiments comparing alternate messages for Cobol syntax checking, text editors, and job control language.

10. Woody, C.A., et al. A subject-content oriented retriever for processing information on-line (SCORPIO). 1977 Nat. Comptr. Conf., Vol. 46, AFIPS, Arlington, Va., 1977, pp. 449–454.