The importance of percent-done progress indicators for computer-human interfaces

Article *in* ACM SIGCHI Bulletin · April 1985

DOI: 10.1145/317456.317459

1 author:

Brad A. Myers
Carnegie Mellon University
509 PUBLICATIONS **21,711** CITATIONS

The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces

Brad A. Myers

Department of Computer Science
University of Toronto
Toronto, Ontario, M5S 1A4
Canada

ABSTRACT

A "percent-done progress indicator" is a graphical technique which allows the user to monitor the progress through the processing of a task. Progress indicators can be displayed on almost all types of output devices, and can be used with many different kinds of programs. Practical experience and formal experiments show that progress indicators are an important and useful user-interface tool, and that they enhance the attractiveness and effectiveness of programs that incorporate them. This paper discusses why progress indicators are important. It includes the results of a formal experiment with progress indicators. One part of the experiment demonstrates that people prefer to have progress indicators. Another part attempted to replicate earlier findings to show that people prefer constant to variable response time in general, and then to show that this effect is reversed with progress indicators, but the results were not statistically significant. In fact, no significant preference for constant response time was shown, contrary to previously published results.

CR Categories and Subject Descriptors: D.2.2 [**Software**]: Tools and Techniques-*User interfaces*; I.3.6 [**Computer Graphics**]: Methodologies and Techniques-*Interaction Techniques, Ergonomics.*

General Terms: Experimentation, Human Factors.

Additional Key Words and Phrases: Progress Indicators, Window Managers, User Interfaces.

1. Introduction.

Unfortunately, there will always be computer programs that cannot be executed instantaneously (fast enough so the user does not notice a delay). Examples of slow processes include compilers, text formatters, file loading from floppy diskettes or other slow devices, file transfers to remote machines or printers, and data base processing. Even with supposedly interactive computer systems that may have "easy-to-use" interfaces with menus, icons or whatever, the user will still be faced with periods when the computer has not finished processing a request.

Percent-done progress indicators are a technique for graphically showing how much of a long task has been completed. They operate like the giant thermometers in charity drives and "fill up" from empty to full as progress is made (see Figure 1). Progress indicators give the user enough information at a quick glance to estimate how much of the task has been completed and when the task will be finished. Many systems currently present a "busy" picture, such as an hour-glass, clock, or Buddha (for "patience"), to show that computation is in progress, but since this is static, it does not indicate how swiftly the program is progressing towards completion or whether the program has crashed or not.

Some systems, such as UNIX* and Accent* (Rashid, 1981), support *multi-processing*, which means that the computer can be performing more than one task at the same time. When multi-processing is coupled with a window management system, such as on the BLIT (Pike, 1983) or PERQ* (Myers, 1984) then the *user* is encouraged to multi-process, i.e. run more than one task at a time. For example, the user might be editing one file while having the system compile another file in the background. Progress indicators can be used in this case to show the progress of *each* process and thereby keep the user informed about the state of the entire environment.

*UNIX is a trademark of AT&T Bell Laboratories. Accent is a trademark of Carnegie-Mellon University. PERQ is a trademark of PERQ Systems Corporation.

Figure 1.
Percent-done thermometer that indicates approximately 70% complete.

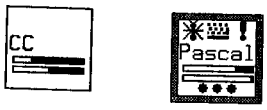

Figure 2.
Two icons from the Sapphire window manager (Myers, 1984), with two progress indicators in each. The first icon shows 25% and 50% complete. The second shows eight pieces of window and process state information including the progress indicators at 75% and 99% complete.

A typical implementation of progress indicators will require that the application programs update them explicitly. This means that if the program crashes, the progress indicator will cease to be updated. Thus, progress indicators also tell the user if a program is still running.

Progress indicators are usually presented graphically rather than shown as a numerical percentage. This has a number of advantages: first, users can more quickly and easily assimilate a graphical display than a textual one (Myers, 1983) when an accurate value is not required. Second, the graphical display implies that only an approximate estimate of the time is available, since exact times can only rarely be determined. Finally, the graphical picture can be displayed in a small space (as in Figure 2) without interfering with other displays on the screen.

2. Implementing Progress Indicators.

Progress indicators can be displayed in a wide variety of formats depending on the display device used, but in every case, there must be some indication of the percent of the entire task that has been completed. For example, on a character terminal, a progress indicator might appear as a series of asterisks along the bottom of the screen, with completion signified by the asterisks reaching the right margin (Figure 3). On a bit-map display, such as those found on personal workstations, progress indicators can be shown as a growing bar (Figure 3), an hour glass filling up, or a clock face with the

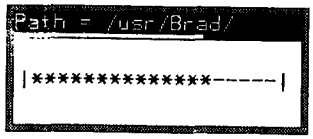

Figure 3.
A window from Sapphire showing a graphical progress indicator in the title line, and a textual progress indicator using characters.

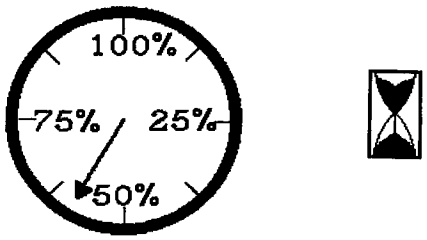

Figure 4.
Two other styles of progress display. The hand on the "clock" face moves clockwise to completion, and the "sand" in the hourglass moves from the top to the bottom.

hands moving (Figure 4). There should clearly be a centralized routine to display the actual pictures so progress for all programs is uniform. In addition, there may be auxiliary routines; for example, one might take a file variable (as in UNIX or PASCAL) and show progress for the percent of the file read.

No matter how progress indicators are displayed, programs will need to be able to calculate percentage information for the indicators. This will clearly be easiest with algorithms that linearly process their input and are then completed. Luckily, a fairly large number of operations fall into this class. Examples include file transfers, program loading, compilation, text processing, etc. These account for a large proportion of the long programs run on many systems. Unfortunately, all of these may also have non-linear parts. For example, a program to be compiled may refer to other programs (such as files that are "imported" or "included") which must also be processed. Also, the "piping" mechanism in UNIX makes it difficult to tell how long the input will be since it may come from another program. This problem might be handled by insuring that all programs in the pipeline are processing their input at approximately the same rate and basing progress on the original data producer (either a file or a program creating the data).

In programs that run multiple passes through data, the progress indicator can be divided into sections for each pass. Since progress is just an approximation anyway, programs may be able to estimate how long they will run based on heuristics or past experience. In addition, if a system supports a hierarchy of programs, for example through command files or scripts, it may be useful to present multiple progress indicators for the same process. For example, the Sapphire window manager (Myers, 1984) has two progress indicators in its icons, one for the current program and another for the entire task (Figure 2).

Figure 5.
The "busy bee" moves randomly around the screen to show that the system is processing (PERQ, 1983).

Figure 6.
An icon from Sapphire with two progress indicators showing random progress. While the application computes, the appropriate indicator flickers.

For programs that simply cannot calculate how long they will be running, a system can provide *random progress*. This can be shown in a number of ways, such as simply printing dots, moving around a "busy bee" (Figure 5), or a constantly changing a pattern (Figure 6). This tells the user that the system is processing his request and has not crashed, even though no information is available to display the percentage completed. A question, however, is whether having some programs with percent-done progress indicators and some with only random progress will be more annoying to users than simply having no progress at all. Experience with POS (PERQ, 1983) suggests that this is not the case.

Progress indicators are not a new idea. For example, Spence (1976) reported that a graphical count-down clock (Figure 4) was used to show the time left to complete a request in a CAD-CAM application. They are also used for file transfer in the Macterminal program on the Apple Macintosh (Williams, 1984). For some reason, however, progress indicators have only rarely been used. Experience with the PERQ POS (PERQ, 1983) and Sapphire (Myers, 1984) systems, which have thoroughly integrated progress indicators with their user interfaces, have suggested that progress indicators are very useful for a wide variety of applications. There is clearly some cost, however, associated with progress indicators in algorithm design and execution time, so it is appropriate to try to determine if they are, in fact, perceived as useful by users. This paper reports the results of a formal experiment that shows that people do, in fact, prefer to have progress indicators.

3. The Experiment.

3.1. Hypotheses.

This experiment was designed to test three hypotheses. The first was simply that people preferred systems with progress indicators. The second was that the progress indicators would be more useful when the response time of the system was variable rather than constant. Earlier experiments, such as Miller (1977) and practical experience (Carbonelle, 1968 and Weisberg, 1984) have shown that users prefer to have predictable, constant response times rather than variable times, even if the average time is shorter for the variable case. A third hypothesis for this experiment was that this effect could be reversed by having progress indicators.

3.2. Method.

A simplified computerized transportation management system was prepared for this experiment which performed simple pattern matching on a data base containing about 100 travel entries. The task was similar to the Miller (1977) experiment. Subjects first read an instruction sheet explaining the task and the commands available, and then they answered eight questions (see Appendix A) using the computerized system, which required about 14 queries to the data base. The questions were on a piece of paper and the answers were written onto a separate answer sheet. The computerized system ran on a PERQ personal workstation (Rosen, 1980) and was designed to be easy to use. To make a query, the subjects filled out the form on the screen (Figure 7), and then hit a key to have the system match it against the data base. Before the results were printed, however, there was a delay that was either a constant 10 seconds, or randomly varied with a uniform distribution from 1 to 17 seconds with an empirical mean of 8.601 seconds. During this delay, a progress indicator may have been shown. After completing all of the queries, the subjects filled out a questionnaire to gauge their feelings about the system. This featured a "semantic differential" scale that attempted to measure the subject's attitude towards the system. There were 10 items with a range of 1 (negative) to 9 (positive); for example: "Anxious...Relaxed" or "Bored...Excited" (see Appendix B). The aim was to measure various aspects of the user's feelings. The dimensions were chosen intuitively. The subject then repeated this process with a different version of the system. The subjects were divided into four groups to determine which versions they used, as follows:

1: Constant time: First "Progress" then "No Progress"
2: Variable time: First "Progress" then "No Progress"
3: No Progress: First "Constant time" then "Variable"
4: Progress: First "Constant time" then "Variable"

Each group was further subdivided so that half got one version first and the other half got that version second. The system randomly assigned subjects to the groups, with the constraint that there would be the same number of people in each group.

After answering the second set of questions, the subject answered the same questionnaire as before to evaluate their opinion of the second version. Finally, the subjects answered a different questionnaire that asked the subjects to explicitly compare the two systems and also asked some background information. All subjects completed both versions and all questionnaires in the same session, and the average times were about 5 minutes to read the instructions, 10 minutes for each version, and 5 minutes for each questionnaire, for a total of 40 minutes per subject.

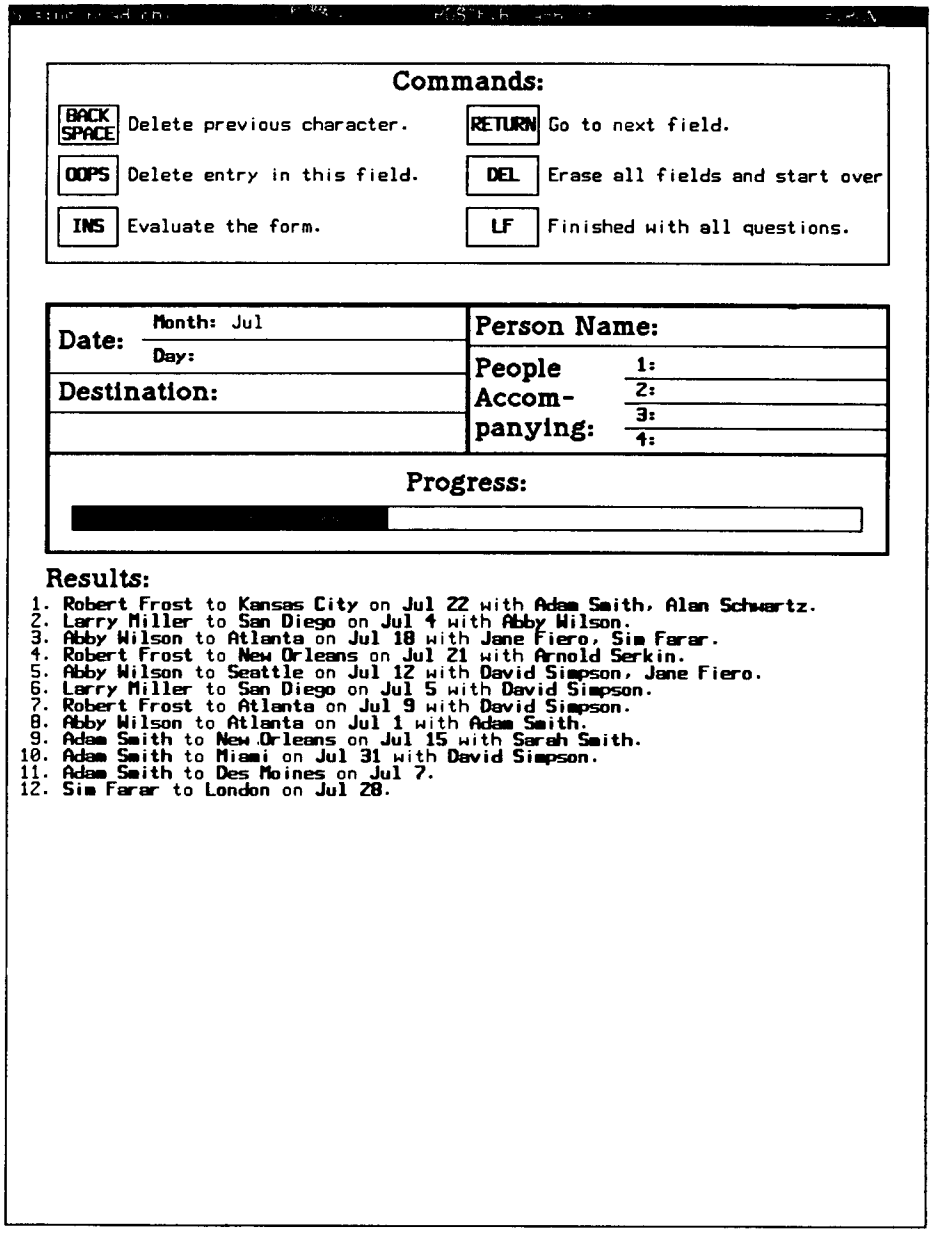

Figure 7.

The screen for the forms-based query system used in the experiment. The subjects typed into the various fields (e.g. "Month", "Person Name") and then hits the INS keyboard key to have the query processed.

	1st 2nd	V: P V: NP	C: P C: NP	V: NP V: P	C: NP C: P	NP: C NP: V	P: C P: V	NP: V NP: C	P: V P: C
Mean Score	1st	6.250	5.867	5.917	5.767	5.233	6.067	6.367	6.216
	2nd	4.317	5.233	6.067	5.733	5.033	5.383	6.683	5.950

Table 1.

Unnormalized means of the scores on the semantic differential (1=negative, 9=positive) on each of the 16 versions. Each column represents one group of 6 subjects (each subject used two versions). The top entry in each pair is the first version the subject used, and the bottom is the second. Code: P=Progress indicator, NP=No progress, V=Variable time, C=Constant time.

Source	df	Sum of Squares	Mean Square	F Value	pr > F
Subject	47	10713.5729	227.9484	5.78	0.0001
Var-Const	1	65.3333	65.3333	1.66	0.2046
Prog-Not	1	540.0208	540.0208	13.70	0.0006
First-Sec	1	404.2604	404.2604	10.26	0.0025
V.C.*P.N.	1	94.0104	94.0104	2.39	0.1296
Error	44	1733.8750	39.4063		

Table 2.

Evaluation of significance of semantic differential scores.

3.3. Population.

Forty-eight subjects were tested, most of whom were computer science graduate students, but about one fifth were computer novices. All subjects were unpaid volunteers.

3.4. Results.

The subjects did not have any trouble learning or using the system. All versions were rated as easy to use, based on the favorable comments from subjects and the fact that there were few errors.

Table 1 gives the means for the subjects' score on the semantic differential scale based on the different versions. These data were fed into the SAS statistics program (SAS, 1984) which generated the data for Table 2. Table 2 demonstrates that the difference for progress indicators versus no progress indicators is highly significant (pr = 0.0006). Also, results from the comparison questionnaire show that 86.1% percent of the subjects liked progress indicators, and the mean rating for them was 2.94, where 1 means "Very Useful" and 9 means "Useless, Annoying." Other significant results are that there was a substantial difference between subjects, and the first version people used was rated higher than the second.

An interesting result is that there is no statistically significant preference for the version with constant response time over the version with variable response time (pr = 0.2046). Even when only the versions without progress indicators are considered, the result is still not significant (pr = 0.1497). Figure 8 shows the average scores for variable and constant time for versions with progress and no progress.

	1st 2nd	V: P V: NP	C: P C: NP	V: NP V: P	C: NP C: P	NP: C NP: V	P: C P: V	NP: V NP: C	P: V P: C
Mean Variability	1st	4.16	7.83	7.33	7.83	6.33	8.33	6.33	6.50
	2nd	4.33	6.67	4.00	7.00	6.33	3.17	6.83	7.66

Table 3.

Unnormalized means of the perceived variability of the response time (1=very variable, 9=constant time) on each of the 16 versions. Format is the same as Table 1. Code: P=Progress indicator, NP=No progress, V=Variable time, C=Constant time.

Source	df	Sum of Squares	Mean Square	F Value	pr > F
Subject	47	197.1667	5.507	1.45	0.1079
Var-Const	1	35.0208	35.0208	12.11	0.0011
Prog-No	1	7.5208	7.5208	2.60	0.1140
First-Sec	1	28.1667	28.1667	9.74	0.0032
V.C.*P.N.	1	35.0417	35.0417	12.12	0.0011
Error	44	127.2500	2.8920		

Table 4.

Evaluation of significance of the subjects' rating of the variability perceived in their versions of the system.

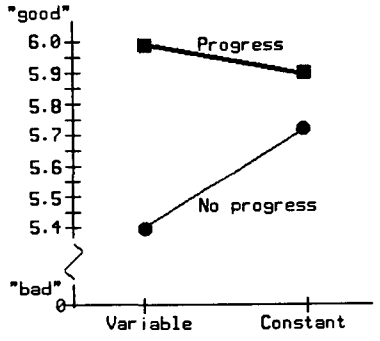

Figure 8.

Graph of the mean scores on the semantic differential for progress versus no progress and variable versus constant time. The sixteen versions were summed into these four groups. It is interesting that constant time is rated better (higher) than variable time when there is no progress, but this affect is reversed when there is progress, as hypothesized. This affect, unfortunately, is not statistically significant.

Tables 3 and 4 show the results when the subjects' evaluation of the variableness of the version is evaluated. The significant result here is that there is a high correlation between having a progress indicator and correctly rating the variability.

3.5. Discussion of Experimental Results.

Clearly, this experiment strongly supports the hypothesis that users prefer progress indicators. This result is statistically significant at pr = 0.0006, which means that there is only 6 chances out of 10,000 that this effect would happen by random chance.

Unfortunately, the last two hypotheses, that progress indicators would affect their feelings towards variable response times, and that variable response times would affect their feelings towards progress indicators, were not supported. It is interesting to note, however, that subjects prefered constant time (mean = 5.73) over variable time (5.41) without progress bars, but prefered variable (5.98) over constant (5.90) with progress bars as hypothesized (see Figure 8). Since this difference is not statistically significant, the experiment failed to duplicate the results of earlier experiments such as Miller's (1977) which said that subjects should favor constant time over variable time, at least without progress indicators.

By observing the subjects perform the experiment, it was clear that when the progress indicator is present, the subjects tended to watch it on the screen since they had no other task to do. Without a progress indicator, however, the subjects apparently got bored with the screen and looked around the room or at the questions or instruction sheet. When the answers appeared on the screen, the subjects would notice this in their peripheral vision, and then look up. This is supported by the data in Tables 3 and 4 which show that the subjects rated the variability of the constant and variable versions the same without progress indicators, but there is a significant difference when progress indicators were present (correlation V.C. with P.N. at pr = 0.0011 in Table 4).

This calls into question, therefore, the general applicability of the earlier experiments; variable time is not always perceived as worse than constant time. In the Miller experiment, for example, the variability was apparently in the rate at which characters were displayed, which is an entirely different situation from the one tested here. Clearly, if the degree of variability is very low, the system will seem constant, and if it is very high, *e.g.* 1 second to 1 hour, it will be unacceptable no matter what the mean is. An experiment to investigate the range of variability that is acceptable, with and without progress indicators, and under various wait conditions, would be interesting. Another approach would be to try to insure that the subjects paid more attention to the screen, possibly by making the test timed and having a very faint signal when answers were ready, or by having the questions displayed on the screen so there is no paper to look at.

Another interesting result is that subjects overall had a lower opinion of the second version they used than the first version which is statistically significant (pr = 0.0025). This suggests that people got bored with the system (due, no doubt, to the long response times) and were annoyed at having to use it again. Since the experiment controlled for this affect, however, it does not bias the normalized results.

4. Interpretation of Advantages of Progress Indicators.

This section attempts to propose some explanations for why people prefer the versions with progress indicators. Since applications will typically not start showing the progress indicators until they have parsed and understood a command, progress indicators provide the following important messages listed by Miller (1968):

> the user knows (a) his request has been listened to, (b) his request has been accepted, (c) that an interpretation of his request has been made, and (d) that the system is now busy trying to provide him with an answer.

Progress bars are important for novice users since they are likely to believe that everything on the computer should operate quickly, and therefore is more likely to panic (Foley, 1974) and think that the computer has crashed if it does not provide feedback while the computer is working. Although experts typically have a feel for how long most tasks will take, they should also benefit from progress indicators. Experts will be more likely to run multiple tasks in parallel since their time is valuable, but most people find it difficult to keep track of what is happening when multi-processing. Progress indicators help users plan and monitor the various tasks so their time can be more effectively used. An interesting experiment would be to attempt to measure the affect of progress indicators in a multi-processing environment.

Another reason people may prefer systems with progress indicators is that they rarely like to sit idle and waste time. Therefore, any "wait" time is annoying. There are many examples of this effect in areas other than computers, *e.g.* waiting "on hold" on the phone. When people do not know how long the wait will be, it is impossible for them to schedule a different task of the appropriate duration, or even to relax effectively. This tends to raise the level of tension while waiting for completion. If there is an indication of progress, however, or if the user knows *a priori* how long the task will take, then the time can be used in some productive manner. This lowering of the users' anxiety is an important benefit of progress indicators. Therefore, an alternative to progress indicators might be an actual number or an analog display of the actual time left until the task is completed, if this can be estimated.

5. Conclusion.

Percent done progress indicators appear to be an important user interface tool that helps users in a number of ways. They help novices feel better about the system by showing that a command has been accepted and the task is progressing successfully. They are also useful for experienced users since they provide enough information to allow them to estimate completion times and therefore plan their time more effectively. This is especially important with multiprocessing systems with windows. The experimental evidence presented here demonstrates that systems with progress indicators are prefered by users. This indicates that the benefits of progress indicators are probably sufficient to warrant the extra cost in computation and implementation required to include them in future systems.

Appendix A.

The questions that the subjects were asked to answer using the query system in the experiment included:

Who wanted to go to Fresno in December?

Who wanted to go to Santa Barbara on August 8?

How many requests for trips to Oakland are there in the data base?

List the dates of travel for requests that William Powers made, where he requested Ned Maxwell to also travel.

If the people who accompanied Scott Derry to Oakland in May, how many trips did each request for themselves in that month?

Which of Walter Sedlak, Rosa Velasco, or William Powers booked the most trips?

Appendix B.

The following is the semantic differential scale used in the questionnaire:

This version of the program made me feel:

Sad	1	2	3	4	5	6	7	8	9	Happy
Anxious	1	2	3	4	5	6	7	8	9	Relaxed
Impatient	1	2	3	4	5	6	7	8	9	Patient
Annoyed	1	2	3	4	5	6	7	8	9	Calm
Tired	1	2	3	4	5	6	7	8	9	Energetic
Uncomfortable	1	2	3	4	5	6	7	8	9	Comfortable
Helpless	1	2	3	4	5	6	7	8	9	Powerful
Bored	1	2	3	4	5	6	7	8	9	Excited
Tense	1	2	3	4	5	6	7	8	9	At Ease
Confused	1	2	3	4	5	6	7	8	9	Confident

ACKNOWLEDGEMENTS

I would especially like to thank William Buxton for extensive help in preparing this paper and the experiment. I would also like to thank the many volunteers who participated in the experiment. For help and support with this paper, I would like to thank my wife, Bernita Myers, Neville Moray, Alain Fournier, and many others at the University of Toronto and PERQ Systems Corporation. The research described in this paper was partially funded by the National Science and Engineering Research Council (NSERC) of Canada.

References

Carbonelle, Jaime, Elkind, Jerome I., and Nickerson, Raymond S. (1968). On the Psychological Importance of Time in a Time Sharing System. *Human Factors* **10(2)**, April 1968. 135-142.

Foley, James D. (1974). The Art of Natural Graphic Man-Machine Conversation. *Proceedings of the IEEE* **62(4)**, April 1974. 462-471.

Miller, Lawrence H. (1977). A Study in Man-Machine Interaction. *Proceedings of the National Computer Conference* **46**. AFIPS Press, 1977. 409-421.

Miller, Robert B. (1968). Response Time in Man-Computer Conversational Transactions. *Proceedings Fall Joint Computer Conference* **33(part 1)**. AFIPS Press, 1968. 267-277.

Myers, Brad A. (1983). Incense: A System for Displaying Data Structures. *Computer Graphics: SIGGRAPH '83 Conference Proceedings* **17(3)**, July 1983. 115-125.

Myers, Brad A. (1984). The User Interface for Sapphire: A Screen Allocation Package Providing Helpful Icons and Rectangular Environments. *IEEE Computer Graphics and Applications*. **4(12)**, December 1984. 13-23.

PERQ POS Operating System Manual. (1983). *PERQ System Software Reference Manual, POS Version G3*. PERQ Systems Corporation, Pittsburgh, PA. May 1983.

Pike, Rob. (1983). Graphics in Overlapping Bitmap Layers. *ACM Transactions on Graphics*. **2(2)**, April 1983. 135-160.

Rashid, R. and Robertson, G. (1981). Accent: A Communication Oriented Network Operating System Kernel. *Proceedings of the 8th Symposium on Operating Systems Principles*. Asilomar, CA, December 1981. 64-75.

Rosen, Brian. (1980). PERQ: A Commercially Available Personal Scientific Computer. *IEEE CompCon Digest*. Spring 1980.

SAS. (1984). *The Statistical Analysis System*, Release 82.4. SAS Institute Inc. SAS Circle, PO Box 8000, Cary, N.C. 27511-8000.

Spence, Robert. (1976). Human Factors in Interactive Graphics. *Computer Aided Design* **8(1)**, January 1976. 49-53.

Weisberg, David E. (1984). The Impact of Network System Architecture on CAD/CAM Productivity. *IEEE Computer Graphics and Applications* **4(8)**, August 1984. 36-40.

Williams, Gregg. (1984). The Apple Macintosh Computer. *Byte Magazine*. **9(2)**, February 1984. 30-54.