Usability of an interface for sketching graphs

a study in Human-Computer Interaction

by Robin Stewart
January 27, 2005
COGS 99
Winter Study Project

Introduction

Graph Sketcher is a drawing application I've designed for plotting data and sketching out graphs. It's designed to be fast, flexible, and immediately understandable. In general, it aims to make the process of sketching graphs on the computer at least as easy as doing it by hand. I built the software from the ground up over a period of two years, accumulating approximately 12,000 lines of code along with interface layouts, icons, and documentation. A back-of-the-envelope calculation (Wheeler 2005) estimates that it would cost over $300,000 to pay professional software engineers to develop it; I put in the time primarily as a learning experience. The result is a piece of software that is both unique and powerful, with enough complexity to give rise to substantial questions about its interface and usability.

I created the software with a range of design and usability principles in mind. For example, Norman (1988) emphasizes the importance of "**natural mappings**" between the desired state of the system and the actions required for the user to achieve that state. In Graph Sketcher, clicking and dragging lines and fills across the graph seems like the most natural analogue to sketching the graph by hand. Norman also emphasizes the need to make controls **visible** and understandable and to clearly show the results of all actions. This is accomplished in a myriad ways in Graph Sketcher: all object properties are shown in the "properties window"; wherever possible, changes are updated in real time; and visual cues indicate which objects will be acted upon. A major goal of my study was to

find out to what extent these elements were actually natural and visible to real-life users, and how they could be made more so.

Raskin (2000) focuses more specifically on software interfaces and gives an interesting and pertinent discussion of "**modes**". Modes refer to the situation where the same actions produce different results depending on the state of the system. The problem is that users tend to forget which mode the system is in (even if the mode is clearly visible), and thus execute actions with unintended results. On the other hand, modes tend to lead to less cluttered interfaces, allow fewer controls, and aid visibility of options. With Graph Sketcher, I faced the dilemma of whether to include a toolbar and if so, how it should operate. The major benefit of the toolbar is that it tells users what they can do ("draw", "fill", "text") and how to do it (click the tool, click the graph). But the major drawback is all of the "mode errors" that users are bound to make as they automatically click and drag without thinking about the current mode.

Raskin recommends substituting modes with "**quasimodes**": temporary modes which are established by holding down a modifier key. Users tend to remember that they have established a quasimode because the act of physically holding down a key continually sends neural messages to the brain. I decided on a compromise for Graph Sketcher: retain the toolbar and the existence of multiple modes, but also support modifier keys which turn these into less error-prone quasimodes. I also tried to alert users to the existence of this method of operation with several types of included documentation.

Finding out how real users would interact with these modes was one of the main research goals.

In contrast with these somewhat abstract guidelines, Apple's Human Interface Guidelines (2005) are very specific. They reflect the known principles of human cognitive ability, but the emphasis is on fostering a **consistent** user experience across all Macintosh programs. This follows the principle that whenever new technology requires users to learn new paradigms (such as saving to disk or moving a cursor), they should only have to learn them once. Thus Apple's guidelines lay out a set of standard commands, keyboard shortcuts, window design rules, and so on – which I tried to adhere to as closely as possible. Thus another main goal of my study was to see if Macintosh users who are used to these particular standards would indeed have an easier time using the software. More broadly, I was interested to find out what methods different users would employ to try to accomplish the various graph sketching tasks.

Other, more self-explanatory goals were to gain experience running a usability test and to generate specific ideas for how to improve the software.

Method

I recruited 16 Williams College students to participate, all of whom were third- or fourth-year students and knew me personally. Some of the participants had heard me describe my efforts in developing the software, but none had ever used it. Each participant was

scheduled to come to an individual "usability session" in a small and quiet computer lab. Each session consisted of an introduction, a standard set of tasks for the user to attempt to complete, and a debriefing questionnaire.

I began each session by reading a script which briefly outlined the purpose of the study, the role of the participant, and the nature of the tasks they would perform. Each participant was told that they should speak their thoughts aloud but that I would not be able to help them complete the tasks. After they signed the consent form, I told each participant to follow the instructions carefully but to remember that they were only creating "sketches" – making the graph "look right" was the most important thing.

During the task part of each session, I sat next to and slightly behind the user, taking notes on a laptop computer. There were 3 groups of tasks, all printed on paper; the same packet was given to each participant. The first task was to "draw a line from the point (5, 0) to the point (15, 10)" and then remove it. The second task was to re-create a hand-drawn graph with colored lines, labels, and a filled-in area (Figure 1). The third task involved plotting data from an Excel file into Graph Sketcher. I took notes on any interesting behavior of the user and the software. For example, I recorded mode errors, points where the user experienced difficulties or quick success, bugs in the software, and system crashes.

My general rule was to not say anything to the subject while they worked on the tasks, but there were several major exceptions:

1. If the user stopped speaking their thoughts to the point that I wasn't sure what part of the task they were working on, I asked "what are you thinking?" or "what are you working on?"
2. If the user asked me a question intended to clarify the instructions printed in the task packet, I generally answered them. For example: "is it possible to rotate the text?" Yes. "Should I scale the axes according to my own data?" Yes.
3. If the software crashed during the session, I stepped in and told the user how to proceed. Usually the tasks were in a very early stage when this happened, so I told users to simply relaunch the program and start the current task again.
4. If the user was about to make an action that I knew would cause the program to crash, I interrupted to tell them not to do it.
5. If the user tried an action that *should* have worked but didn't because of a known bug, I interrupted and told them how to use a workaround. The rationale behind this was that their action is supposed to have worked, so it would be a waste of time to just let the user struggle.
6. If the user was struggling with one detail for unreasonably long and generally going in circles, I told them to just skip that detail in the interest of time. Again, the rationale was that letting the user move on would be a more productive use of time.

I also set up the software to monitor certain quantitative data about each session, such as the number of mouse clicks and menu accesses. This data was saved to a log file

periodically. Unfortunately, whenever the program crashed, some of the data was lost. Thus there is only valid data for a subset of the study participants.

At the end of each session I asked the participant if there were any questions they were dying to ask or features that they wanted me to demonstrate. After a few minutes, I asked them to fill out a questionnaire including both open-ended and circle-the-best-choice questions. The questions were intended to assess users' enjoyment of the program, find correlations between performance in the study and background knowledge, and solicit feedback.

Last and probably most importantly, I modified the software three times during the course of the study in response to some of the major problems subjects were having. All of the modifications were relatively easy to make, but sometimes quite substantial in nature. One major change involved the manipulation of text labels: I made it harder to accidentally create labels, and easier to stop editing labels. Late in the study I modified the toolbar, which is discussed below. Smaller but important changes include altering the behavior of the "Connect Points" button to the behavior that most users seemed to expect; and adding an "Import..." menu item to instruct users how to import data (very useful for the third task). In addition, I fixed five or six small bugs that subjects had uncovered.

Results and Discussion

By far the most interesting results came from qualitative observation of the test subjects. Overall, most people had a fairly easy time figuring out how to use the software and almost all comments were positive. Many remarked that though it had been difficult to figure out the software without any help, now that they understood it they could easily put together new graphs. These comments were verified by the fact that subjects got exponentially faster as they continued to use the program. Although every participant took a completely unique path in trying to accomplish the tasks – and seven did not complete all tasks given the allotted time – there were several major (and surprising) themes that emerged.

The Toolbar and Mode Errors

The clearest and most widespread phenomenon was that virtually all participants using the original toolbar version of the software made at least one mode error during their session. They would draw a line when meaning to move a label; create a label when they meant to select an axis; scale an axis when they meant to draw a line. On one occasion the user never noticed that she was making a mode error and instead concluded that the software did not allow moving text labels around. But usually subjects would figure out the problem within five or ten seconds – and matching perfectly with Norman's (1988) observations, they did not generally blame the software but rather chided themselves, muttering things like "oh, I'm dumb."

Despite the modifier key shortcuts being prominently advertised in the “Really Quick Reference” box near the top of the screen, almost no one ever used them to enter tool modes. When asked why (at the conclusion of a session), some users said it was "easier to just click in the toolbar" rather than hold down a key. But was it easier merely because they were used to clicking toolbars? Or are toolbars inherently more understandable? Perhaps more analogous to picking up another tool in real life? Oddly enough, the only certainties are that clicking the toolbar is much more prone to mode errors and moreover is a lot slower than pressing a modifier key.

These results emphasize the common dilemma of whether or not to listen to users. Do they really know what’s good for them? One could argue that by not including the graphical toolbar, users would be forced to learn the modifier key method, and in the end would be thankful because of their improved speed and lack of mode errors. On the other hand, if users were given a choice between two identical versions of Graph Sketcher except that one included the toolbar and the other didn’t, it seems pretty clear that users would prefer the one with the toolbar “feature”.

As a compromise, I decided to retain the toolbar and refine its interface in several ways. First, I removed the “functionality” of creating new text labels after every double-click – a “feature” which had ended up resulting in many more undesired text labels than desired ones. Second, I programmed the toolbar to switch back to “Modify” mode after any line, fill, or text label had been created. This followed the observation that most often, users

wanted to modify the most recently drawn item before adding something else. Four subjects had the benefit of using this new version of the software, and between the four of them I noticed *no* mode errors, leading me to believe that the modification was a resounding success.

Other Qualitative Results

In general, it seemed that users did not expect the program to be as easy to use as it actually is. For instance, some users started out by creating every line via entering coordinates in the properties window, and discovered only by accident that they could draw lines simply by dragging from place to place. Few, if any, subjects realized that almost all operations could in fact be done without any recourse to the properties window. And few subjects spent enough time looking through the menus to discover useful time-saving features such as "Select all points". I suspect that this is partly due to the fact that I was asking them to do specific tasks; subjects probably did not feel at liberty to just explore the program. Indeed, many reported that they didn't think they were "allowed" to even use the Help menu. Still, it is likely that users in real life situations would also be under time pressure and would thus ironically fail to uncover some of the shortcuts. This is a problem which I'm not sure how to address.

Another interesting result was that copying data directly from Excel into the graph window was too much of a stretch for virtually all participants. Instead, users found the list of points drawer (which has the same layout as a spreadsheet) and pasted into there. This was a complete surprise to me, but seems fairly obvious in retrospect. However, I

think it has a lot to do with user perceptions of the current limitations of computers. One can't usually copy and paste between disparate data structures, even when there is a meaningful connection. Happily, almost every subject who made it to the third task did try the copy and paste technique. They would say things like, "I can't imagine I can just copy and paste... but I guess I'll give it a try."

One unexpected and (practically speaking) unfixable issue that many users encountered came in the form of the color picker window. Apple has designed a wonderful, multipurpose color picker which includes every conceivable way of choosing colors. The only problem is that the default way starts out by displaying a large, black circle. To find actual colors, the user has to either drag an unlabelled slider or choose another method of color picking, via again-unlabelled icons. Some subjects spent 20 or 30 seconds trying to figure this out before either succeeding or giving up altogether! Here the good news is that once a new method of color picking has been selected, it shows up by default every time, so those 20 seconds are a one-time cost.

Quantitative Results

The data gathered from questionnaires and logging user actions were not nearly as interesting or conclusive as my qualitative findings. Most of the measurements in the log data were limited in ways that rendered them fairly useless for analysis. The measure that does seem useful is the number of "**steps**" it took each user to complete each task. A step is defined by any action or set of actions that would be undone as a group if the "Undo" menu item is used. The nice thing about this measurement is that it includes

almost every part of the process of working with the software, and it is related to the amount of time the user must spend on a task. On average, it took subjects a total of about 50 steps to complete tasks one and two, and less than 20 steps to complete task three; but there was a large amount of variation.

Unfortunately, there were not enough Macintosh users in the study to find any substantive differences between the performance of Mac users versus Windows users. For instance, it took Mac users approximately the same number of steps to complete the tasks as it did Windows users. One hypothesis to explain this result is that Mac users might experiment more (generating excess steps) but also figure out the correct methods faster (lowering the number of steps) than their Windows counterparts. Qualitatively speaking, Mac users seemed more confident using the program, but made a similar number of mistakes.

Similarly, the general trend in the questionnaire data is that there are no clear correlations between any factors. For instance, the departments which users had taken classes in seemed to have no bearing on enjoyment of the program or number of steps needed to complete the tasks. The amount of experience users had had with the various software packages I queried about showed a similar lack of correlations with any of the other data. This seems to suggest that Graph Sketcher is to some extent a program in its own category, with a unique set of interface paradigms. This could be a bad thing if it confuses users or goes against tried-and-true principles, or a good thing if its paradigms are innovative and easy to learn and use.

Improvements and Impressions

Much of the point of the study was to see if real users would attempt the actions I thought they would try in order to complete the tasks. Watching and recording what types of actions the various participants tried was extremely informative and helped me generate a long list of worthwhile improvements for the software. It seems clear that if a lot of users expect to accomplish a goal by a certain method, that method should probably be set up to actually accomplish that goal. Some of the improvements suggested by user actions include using the shift key for multiple selections, better integration of the properties panel with the graph window; better ways to manipulate the axes without using the properties panel; and better handling of nonsensical axis values.

Overall, the results of the usability study were extremely impressive. A majority of users were able to complete two fairly complex tasks within 20 minutes, having never seen the program before – in some cases, having never even used graphing software before. This is a good indication of a highly intuitive interface and speaks highly of the software's potential as an educational tool. It is also encouraging that almost all participants reported that drawing graphs using Graph Sketcher was at least "somewhat easier" than drawing them by hand; that the program behaved as expected at least "most of the time"; and best of all, that they "enjoyed" using it.

References

Apple Computer, Inc. (2005) *Apple Human Interface Guidelines*. Online: <http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/>

Raskin, Jef. (2000) *The Humane Interface: new directions for designing interactive systems*. Reading, MA: Addison Wesley.

Norman, Donald A. (1988) *The Design of Everyday Things*. New York: Basic Books.

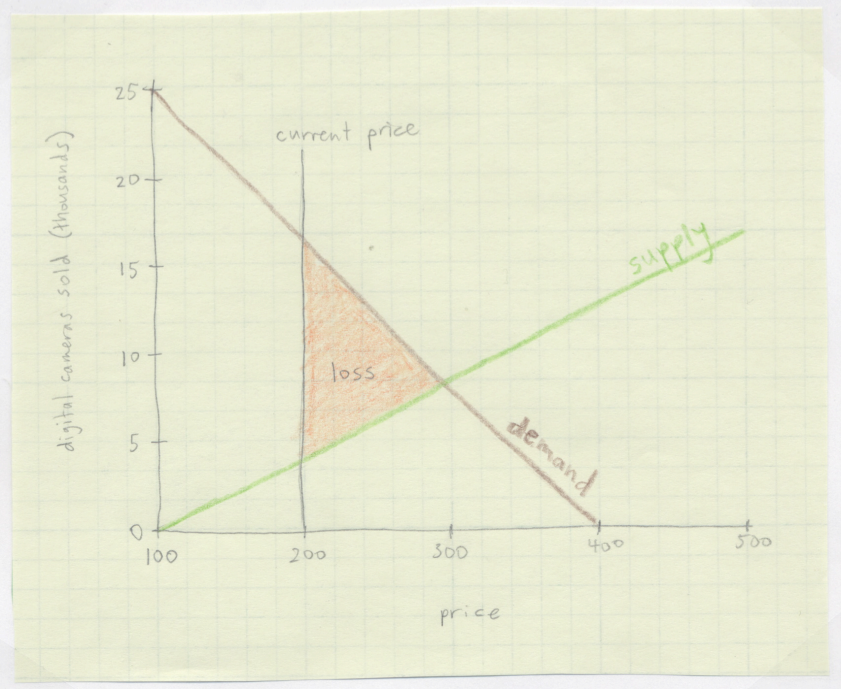
Preece, Jennifer. (2002) *Interaction design: beyond human-computer interaction*. John Wiley & Sons.

Shneiderman, Ben and Catherine Plaisant. (2005) *Designing the User Interface : Strategies for Effective Human-Computer Interaction* (4th Edition). Addison-Wesley.

Wheeler, David A. (2005) Data generated with his freeware program 'SLOCCount' which tallies the number of source lines of code in software projects.

Figure 1.

The hand-drawn version:



Some attempts at re-creating it (task 2):