Paper 165

14 Steps to a Good GUI

John E. Bentley, First Union National Bank, Charlotte, North Carolina

Abstract

Creating a graphical user interface is clearly very different from writing program code, but with SAS/AF®, FRAME Entries, and Screen Control Language the user interface and program code can be almost indistinguishable. This makes SAS application development easier and faster. Programmers, unfortunately, usually do not have visual design training so too often the result is a clumsy interface running powerful code. Luckily, there are straight-forward methodologies for GUI design that can be easily implemented. Drawing primarily on the work of consultant and author Susan Fowler, this paper will present an overview of a 14-step methodology for designing a good GUI. No experience with application development is required although it would be helpful.

Introduction

One of the wonderful things about computers is that you can make one do just about anything with data—enter, extract, manipulate, report, or present it. SAS software excels at empowering users to do all these things. And therein lies a problem: the user must communicate with the computer to accomplish these tasks.

For better or worse, good programmers are not necessarily good communicators and graphical user interfaces are a form of non-verbal communication. It's an example of the left-brain and right-brain dichotomy. Interface design requires visual thinking and the imagination to guess what the entire application will look and "feel" like to the end user. Programmers are necessarily trained in logical and process-based ways of thinking—quite the opposite of what's needed for visual design.

The SAS software used for GUI design—SAS/AF, FRAME Entries, and Screen Control Language—tightly binds the interface and program code to the point where they can be almost indistinguishable. On the one hand, this makes application development easier and faster. On the other hand, often the result is a clumsy interface running powerful code.

Many SAS applications are developed in a few weeks under a form of rapid application development, which for many projects translates as "code and go". Longer, complex applications require a more detailed or structured methodological approach, and there are a number of them to choose from. Methodologies that specifically guide the GUI design process have also been developed, but programmers are usually not familiar with them. This paper will remedy that by presenting a modified version of one methodology.

Overview

In a client-server environment, today's computer-human communication paradigm is based on an intent-driven graphical interface, as opposed to the text-based document orientation still common on mainframe computers. The intent is to allow *someone* to do *something* using the computer. Staying focused on the end-user's intention is often the difference between getting the interface right and having to redesign and re-implement sections of the application.

One way to stay focused is to adopt a "drill-down" method of GUI design. First, you do the standard software application project preparation work—pick a design team, list the user's tasks, and come up with a set of conceptual models of how those tasks are accomplished. Then, using low-level prototype testing, select one model that will guide further development.

From the prototyped model, finalize the home window. Then create the secondary windows, fields, dialog boxes, menus, icons, and control buttons. Lastly create the messages, help files, and other text users will need to keep on track. Throughout the process, test your work with various usability tests and evaluations. (Remember: until your work is user tested it's only an educated guess about what the users need/want.)

Types of applications

It is important to understand that there are different types of applications and that these applications work best with differing degrees of "density"—the amount of white space on the screen. Although the density of the windows is determined in part by the type of application, prototype testing by the users will be important.

Most SAS applications will be ring, core, or sovereign types. SAS/AF Frame is not conducive to designing daemonic, parasitic, and transient application types. Fowler (1998, pp. 241-244) gives a good overview of applications types.

Ring and Core Applications

Ring applications are designed to help users do specific activities in areas with which they are not very familiar. A client address lookup application may be a ring application. Core applications, on the other hand, enhance the user's core competencies. An example is one that extracts data for a statistician to analyze.

Ring applications should be less dense—have more white space—than core applications, and should have more detailed help capabilities. Experience shows that ring applications can be up to 70% dense and core applications can be up to 80% dense. (Spool, 1996.) Since users work with ring applications only occasionally, online help and messaging are of heightened importance there.

Sovereign Applications

A sovereign application is "the only one on the screen" and users work with it for a long period of time. Although each user will of course initially be a novice, it is assumed that they will quickly become an experienced user due to the amount of time devoted to working with the application. Sovereign applications, therefore, should be designed with experienced users in mind. Speed and power should not be sacrificed for a clumsier but easier-to-learn interface. The interface should be dense and have as many keyboard shortcuts, pop-up menus, and drag-and-drops as users can handle. (Galitz, 1997.)

A 14-Step Methodology

1. Pick a design team.

2. Do task analysis and create user profiles.

3. Find a good conceptual model by brainstorming and then do low-level prototyping of the best ones.

4. Finalize the look of the home or main window and the secondary windows.

5. Define the application structure—the menus.
6. Create the task paths with secondary windows and dialog boxes.
7. Create the input fields and data entry areas, which can be screens, windows, or fields.
8. Brainstorm for a consistent look for the icons and buttons.
9. Name all the buttons and define their purpose.
10. Create the buttons, check boxes, and lists.
11. Define data displays—tables, charts, graphs, etc.
12. Create the shortcuts: toolbars, pop-up menus, and option choices.
13. Write error, alert, and status messages, plus on-line help.
14. Review what you've done and prepare for the next release.

1. Pick a Design Team

Notice the operative word here: Team. Interface design is not something to be done by one person alone in a cubicle. It requires a team of people. Table 1 lists the team members and why each is important. Not all projects need a different person in each role, but each role should be assigned.

Table 1: GUI Design Team Members

Role	Importance
Artist/Designer	Identifies the correct visual style and sensible graphics. You might be surprised who has visual design skills: the receptionist, technical writer, or business analyst.
Application Developer(s)	Knows what's possible, from a coding standpoint.
End-User(s)	Always thinking "how will this make my job easier." Involving real users early is the key. (If you do not get a user to help design the app, what makes you think they'll use it?) Do not rely on supervisors to define what their users need.
Facilitator	Tracks schedules, issues, and decisions and keeps everything moving forward. Tip: After each meeting, review what was accomplished to make sure nothing was missed. Another tip: Include the end-user/client in status report distribution lists.
QA Expert	Knows what will make an application easy or difficult to support and maintain. At the very least, use a developer/ programmer not involved in the coding.
Technical Writer	A good stand-in for a novice user. Knows how to view functionality from a user's perspective. Critical point: document as you go.
Trainer	Knows what features and functionality users like and dislike. Knows what confuses them. Trainers should have a wealth of knowledge based on witnessing user's first encounters with other applications.

On a major software application, it will be useful to include some additional team members.

Table 2: Optional GUI Design Team Members

Role	Importance
Recorder	Specifically designated to take minutes of meetings and keep track of all project documentation.
Behavioral scientist or cognitive psychologist	Understands how the human mind works and how people perceive and do things.
Sales and account representatives or the business unit liaison	Can provide insight on what features and functionality it will take to "sell" the application to the users.

2. Do task analysis and create user profiles.

Common sense says that before you start designing it would be wise to figure out whom is going to use the application and what they are going to use it for. Too often, however, this is given only cursory attention. While it's not necessary to go overboard, it is critical to get a solid understanding of what users need, want, and expect.

The reason to do task analysis and create user profiles is not to fill out the documentation file but to replace IT staff perception, opinion, impression, and argument with real, honest input from the users. Later design meetings will include brainstorming sessions in which perception and opinion were formed by this user input. When strongly-held differences arise during design meetings, and hopefully they will, the facilitator assigns team members to resolve them via more research and user interviews.

Documenting the results of task analysis and user interviews is important. It provides the audit trail that provides input to later decisions and gives clear consistent information to management and other interested parties, including the users themselves.

One older approach to design specification is the "waterfall approach" wherein all aspects of the application are specified in advance and are not deviated from during development. Most projects, however, cannot use this highly structured approach. Most client-server application development will be accomplished via an iterative process based on a form of the 8-stage application development methodology or a rapid application development methodology (RAD). DeGrace and Stahl (1990) cover earlier application development approaches, and the University of California-Davis web site (see bibliography) has an excellent series of documents on the 8-stage methodology and RAD.

Two approaches for doing task analysis are use cases (Constantine, 1995) and task scenarios (Lewis and Reiman, 1993). For an overview of user analysis, see Rubin (1994) and Dumas and Redish (1994).

If the project is to re-work an existing application,

- Do not take a bad workflow design as given. Insist on re-engineering the process first, then automate it.
- Eliminate unnecessary constraints and restrictions, and throw away unhelpful or unneeded choices.
- Give the users as much freedom as they can use, not as much as is technologically possible.

3. Find a good conceptual model by brainstorming and then do low-level prototyping of the best ones.

“The conceptual model or metaphor of a software user interface is the means by which it communicates the software’s underlying operations and functionality to a user.” (Rubin, 1966, p. 130). An example of a conceptual model for an EIS might be a desk file drawer showing a series of labeled folder tabs. Clicking on one of those would open a specific report or graphic.

For many projects the conceptual model may be apparent or preordained. An application that does data extraction, manipulation, and file creation should look like others that the user group is familiar with. When a task is automated for the first time or moved from one platform to another (IBM MVS mainframe to HP-UX client-server) you do not necessarily know what the application should look like. In these cases brainstorming sessions are needed.

To find the right conceptual model, numerous possibilities need to be identified and refined via prototyping. Why? The conceptual model has a primary effect on usability, and users have specific experience-based look and feel expectations. If the application’s conceptual model closely coincides with the user’s mental model, then the application will be easier to learn and use. If not, the user will have difficulty moving beyond the novice level.

To identify possible models, the team must spend a few hours brainstorming. The facilitator must make sure everyone understands that no idea is wrong or silly, and keeps pushing the group to come up with ideas. Obvious models are identified early on, and the richer, more provocative models appear only later.

Models are eliminated or combined based on:

- How well they can be extended to address the specific user need. A bookshelf model may be appropriate for an application presenting preexisting tabular or graphic reports but not for an ad-hoc data analysis application.

- Whether the target audience—the users—will readily understand them. For example, a bag of money icon to indicate Savings Accounts may amuse some employees but will baffle others.

To find out which model has the most resonance with users, create low-level prototypes for each one. This is not a full-scale interface, just the most important functions: the home window, two or three menu items, and a couple of dialog boxes. SAS/AF Frame entries and SCL makes developing these prototypes incredibly easy.

Basically, these prototypes do not “do” anything except allow the user to press buttons, provide some input, make a few choices, and see what happens. User testing at this level is to refine the *concept*. You’ve got to have the concept right before moving on.

4. Finalize the look of the home or main window and the secondary windows.

The home window is not an introductory or sign-on screen. The main window should present the conceptual model and at the same time allow the user to do real work. It is the window that the user starts with and, quite often, ends with. The window that appears when SAS/Windows initializes is a home window.

The home window vs. secondary windows

Under Windows 95, any window that isn’t a home window is a secondary window. Secondary windows are derived from the main window and appear on top of or inside the frame of the main window. Structurally they resemble the main window.

SAS/Windows applies a “workspace” paradigm wherein a window holds a set of objects—other windows. This is opposed to the “workbook” paradigm adopted by Excel, for example.

When SAS/Windows initializes the main window appears and by default immediately holds three secondary windows but displays only two. The Log and Program Editor windows are on top of the Output window. As other examples, when you click on the Libraries or Help icon you are opening a secondary window; from the menu bar, choosing Options, Preferences opens a secondary window.

Types of windows

There are three types of windows: form-based data-entry, conversational (also called “interactive”), and inquiry (“read only”).

- Form-based data-entry windows resemble a paper form and quite often the user is entering data from the source document into the computer. For this reason, it’s important that the window resemble the form to the extent possible using abbreviations, scrolling, and/or paging. Design challenges here are to properly use the limited space of the window to organize the data entry fields.

- Conversational windows are most common. The users look at and interact with the window itself. Information is entered or specified and responses are returned. Conversational windows can quickly become crowded and confusing. A key determinant of whether the window is overly complex is to ask users how well they can identify all the screen elements without having them explained.

- Inquiry windows allow users to submit search criteria and then view the results. An inquiry, however, is more than “Show me customers in Miami and their total purchases.” Depending how the search criteria are entered and the results are returned, an inquiry window can quickly come to resemble a magazine page. The keys are layout consistency via a grid system, placement so that the user can quickly find what they are looking for, and proportion of text and graphics—for optimum readability, a line of type should be about thirty-nine characters long.

5. Define the application structure—the menus.

“The magic number is seven, plus or minus two.” George Miller, 1956.

Menus take advantage of our ability to recognize things that jog our memory. Menu bars (both pop-up and pull-down) are designed to provide navigational clues to help users find the command needed. They do this by “chunking” or grouping similar menu items.

Menu design guidelines:

- Restrict menu depth to three levels or fewer. Performance tests have determined that a menu hierarchy with no more than two submenus and eight items per submenu was the best for speed, accuracy, and preference. A menu with six submenus of two items each was the slowest, least accurate, and least preferred.

- Group menu choices logically and on the individual menus group the options in a logical order. A logical order can be alphabetic, based on frequency of use, chronological or order of use, numeric, or some standard.

- Use sensible menu titles and labels. When possible, limit items to one word and show the shortcut, e.g., Ctrl+F, next to it.

- Be consistent in appearance and logic across menus—proofread. Checking for consistency is less time consuming than documenting inconsistency.

- Let users bypass the menus. Menus slow down many expert users so always provide command line capability.

One way to define menus is to divide the team members into two groups. Working from the results of the task analysis, the first group writes the name and description of all the needed menu items on 3x5 index cards. When they finish, the other group organizes the cards into logical groups and names the groups. Then the cards are pasted in group order on the wall and the team discusses and modifies the results. (Fowler, 1998.)

The team's final product is a documented menu map that is prototyped and tested. Adjustments are made based on user input and the prototype-test cycle is continued until all are satisfied.

6. Create the task paths with secondary windows and dialog boxes.

Secondary windows and dialog boxes keep the application running smoothly and direct users through the workflow. Quite often, two task paths can be defined. The first is a main path starting from the home window and including feedback and error checking/correction. The second is an expert user path that provides less feedback and error checking/correction.

Creating task paths is an obvious place to tap user expertise and at the same time gain user support for your application. Both novice and expert users can provide critical input on how they would get from point A to point B and which information logically (to them) should appear in each intervening window. If the project is a re-engineering of an existing application, sit with a user while they step through it and record their comments.

7. Create the data entry areas.

Creating data entry areas should be easy by this point. Again, user participation will be useful in organizing input fields into logical groups and organizing the groups to match the workflow. Quick prototyping tests will validate your work as you go.

There are three types of fields: data entry, required, and protected. All three overlap and share some features. Required fields are a type of data entry field, and a data entry field can change to protected or vice versa depending on the value of some other field.

- Data entry fields allow users to enter and edit data—search criteria, data set names, or system information. A data entry field is not necessarily an input field that requires the user to enter text. It can also take the form of a multiple-select list box, a series of check boxes, or a set of radio buttons.

- Required fields are just what their name implied—users are required to enter valid data. Required fields are identified by their look or label, behavior, or timing of their appearance.

- Protected fields display data but do not allow editing. These fields may show default parameters, system values, already saved values, or values calculated from data entry fields.

8. Brainstorm for a consistent look for the icons and buttons.

Applications use desktop icons to start the program and iconic labels to operate. A form of the game Pictionary is a good team method for brainstorming icons. (Fowler, 1998.)

- In a brainstorming session, all required (and imagined) icons are listed on index cards, with one icon name per card. Required icons should have been identified during task analysis, but based on experience thus far additional ones may have been identified.

- Divide into two teams, and each team designates a starting artist and a recorder. The cards are shuffled and the starting artist from Team 1 picks a card. He or she has five seconds to think and then must begin drawing the icon. The other team members have sixty seconds to guess the concept the artist is trying to communicate.

- If they cannot guess the concept after sixty seconds, the card is put on the bottom of the stack for another try. The sketch is given to the recorder who labels it with the icon name and sets it aside. Team 2 then takes a turn.

- When a Team gets the concept, recorder keeps the sketch with the card and notes what seemed to work and what did not. The process keeps going until all the icons have been sketched and identified, time runs out, or the Team members are burned out.

A case study of the design of Sun Microcomputer's Web-site icons can be found in Nielsen and Sano (1997)

9. Name the pushbuttons and define their purpose.

Some guidelines for naming buttons are:

- Be specific. Instead of labeling a button "Save", add what the user is saving—"Save Specifications" or "Save Data Set".

- Use the user's frame of reference. Remember, the users likely aren't IT professionals. From other applications, users may be comfortable with buttons where "Go" means "go retrieve the record" and "Run" means "run the calculations".

- Be consistent and use the same buttons for the same functions across the application. When functions are similar but slightly different, look at the task. If one window has a "Go" button, should another window have a "Run" button? Possibly, depending on the user's frame of reference.

- Use industry standards whenever possible. Cancel, for example, is used to cancel any unapplied user settings and close the dialog box, not to close dialog boxes in which no changes were made (use Close instead) or as a synonym for Quit or Exit.

10. Create radio buttons, check boxes, and lists.

The label attached to the overall sets created here must unambiguously identify its use. Try to include a verb referring to the action that will be taken.

- When a single choice must be made from a set of possibilities, use a set of radio buttons over a series of check boxes.

- When there are only two choices (“Show Summary”—Yes/No), use a check box that when selected sets the “on” or “yes” state.

- When there are more than a half-dozen possibilities to choose from, use a list box.

Watch for situations where dynamically filled list boxes (either single- or multiple-select) might be more appropriate than radio buttons and vice versa. Also look for places to replace data-entry fields with dynamic list boxes or combo boxes.

Testing these components during prototyping will initially identify which is more useful and appropriate. If possible, early in the testing process you should begin using the customer’s own data. By doing so, you’ll quickly uncover problems there that might not appear using generated test data. A classic example is customer names that are too wide for the list box that was designed.

11. Define data displays—tables, charts, graphs, etc—and output areas.

Data displays traditionally include tables, graphs, cross-tabs, and presentations of the type found in Executive Information Systems. The data can be raw, such as the first 100 records returned by a SQL extract routine, or clean, like the results of a statistical analysis presented in a graph. Output can either be displayed in a window or sent to a separate file or device.

The team should brainstorm to identify needs and opportunities to transform data from one form into another, more easily interpreted form. For example, if the application displays cross-tabulations, add the capability to display the data as a pie chart. If you display a graph, provide the ability for the user to see the underlying numbers. These are the sort of easily implemented “bells and whistles” that users may not have thought of simply because they’ve never seen them anywhere before. It’s a real opportunity to make their eyes light up.

Graphics have over the past few years expanded to include photographs and video clips, and the wave of the future is multimedia—a combination of video, voice, music, sound, illustration, and text.

12. Create the shortcuts: toolbars, pop-up menus, and option choices.

Many shortcuts are defined at the same time that the toolbar labels are defined. Other shortcut areas that should be explored include:

- Keyboard-oriented shortcuts—mnemonics--especially if you know from task analysis that many of the users have touch-typing skills. There are standard rules for assigning mnemonics:

  ✓ Mnemonics are combinations of the Alt key and one case-insensitive character.

  ✓ In pull-down or pop-up menus, the mnemonic must be underlined and the user presses the character. Elsewhere, the Alt key plus the character must be pressed.

  ✓ Assign the five to seven most important options in the entire menu systems first. For these, use the first letter in the word or phrase. If the first letter is already used, use the first or most “interesting” consonant. Use traditional, standardized mnemonic where appropriate--“Open” is always “O”, “Save As” is always “A”.

  ✓ Do not use the same letter more than once on any individual menu or toolbar item. You can use the same letter on different menus, such as a menu and then a submenu.

  ✓ Duplicate menu items should use the same mnemonic regardless of where it appears. For example, if “Save” shows up in three different places, “S” should be used each time.

- Pop-up menus make context-sensitive tools available for expert users.

- Consider adaptive technologies even if your users do not seem to need them now. Technologies designed for partially sighted customers can be very helpful for users with tired eyes after a long day; those designed for deaf customers are ideal for noisy environments.

- Include those options and shortcuts to all printers, plotters, fax machines, and non-SAS data formats that users infrequently use. Use the results of task analysis to be proactive about possible output devices.

13. Write the error, alert, and status messages, plus on-line help.

Messages should be one of the last items addressed in a methodical manner, but they are of critical importance to user acceptance of the application. Everyone has experienced incomprehensible messages or unhelpful messages that left them confused or angered.

During application development, of course, messages will be developed on an as-appropriate basis. The goal now is to review, revise, and expand the messages comprehensively.

A good place to start is to make a list of all current messages or the messages from an earlier release of the application and rewrite them in a goal/action format. For example, if this was the current message:

Create DIR_NAME directory?

A goal/action form of the message would be:

The DIR_NAME directory does not exist.
Do you want to create it?

Writing messages can be a simple group exercise using a white board or blackboard. For a complex application, however, it may be more effective to assign each team member a set of messages to write and then have another team member critique and revise them.

Online help should be prototyped like the other aspects of the application and included in user tests. Abbreviated help items can be quickly written for prototype and usability testing, but most customers will insist that the user acceptance testing include the final online help documentation. SAS/AF supports native Windows help files, and Boggan (1996) and Microsoft (1995) each provide authoring kits for writing help files. Fowler and Stanwick (1995) provide a good overview of on-line help, and Coe (1996) and Hackos (1994) for address testing online documentation for usability.

14. Review what you’ve done and prepare for the next release

In one sense, no software application is ever a “final” version. Software must be modified as the business needs and requirements change, users pick up new skills and expectations, hardware becomes faster and easier to use. Usability testing very likely will indicate that that users have new or unrecognized needs that are not addressed by the current version of the application.

Modifying the application during its development can fill some of these needs, but many will have to wait for the next release in the product cycle. Plan to maintain formal records of unmet needs that are discovered. Help-desk reports may also indicate where changes and enhancements are. Keep in mind that the list of enhancements could lead to a different conceptual model.

About six weeks after the application is released, hold a team review session. Discuss which parts of the design and development process worked and which did not. Talk about ideas for the next release, even if it's only a maintenance release. Maybe even invite in some users and brainstorm some ideas for enhancements.

Remember to keep all documentation—memos, schedules, task analyses, notebooks, index cards, sketches, prototype code, whatever—together and in one place. Even if the current application is never modified or enhanced, this documentation may provide valuable guidance for the next project.

Conclusion

While clearly not applicable for RAD projects, this 14-step methodology provides structure to the GUI design phase of a SAS application development project. By avoiding time lost to mistakes and false starts, it speeds up—not delays—the completion of the project.

With SAS software, it's usually desirable that the GUI design be done at the same time as the program coding, especially for new applications (as opposed to putting a GUI on an existing batch program). This 14-step methodology allows the GUI design and development work to be included in project planning and management tools such as MS Project as a series of tasks distinct from the coding. The better we can decompose tasks, the better we can plan and manage.

Most of the task analysis information needed by the GUI designers is also needed by the application programmers and vice versa, so there's not much extra effort needed there. And because with a SAS application the programmer often is also the GUI designer, there may not any additional programmer resources needed. Finally, much of the recommended GUI prototyping and testing supports and enhances testing the actual program code.

In the words of Mark Smith of Spinnaker Reach Ergonomics Consulting, a GUI designer/programmer should keep one particular thought in mind: "Someone is going to use this thing as part of their job. My job is to make it as easy as possible for that person to get their job done."

References and Bibliography

Bogan, Scott et al. (1996) *Developing Online Help for Windows 95.* New York: International Thomson Computer Press.

Coe, Marlana (1996). *Human Factors for Technical Communicators.* New York: John Wiley & Sons, Inc.

Constantine, Larry (1995). "Essential Modeling: Use Cases for User Interfaces." *Interactions*, April, 34-36.

Dumas, Joseph S. and Janice C. Redish (1994). A Practical Guide to Usability Testing. Norwood, NJ: Ablex Publishing Corp.

Degrace, Peter and Leslie Hulet Stahl (1990). *Wicked Problems, Righteous Solutions. A Catalogue of Modern Software Engineering Paradigms*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Fowler, Susan (1998). *GUI Design Handbook*. New York: McGraw-Hill.

Fowler, Susan and Victor Stanwick (1995). *The GUI Style Guide.* Chestnut Hill, MA: Academic Press Professional.

Galitz, Wilbert O. (1996). *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. New York: John Wiley & Sons, Inc.

Hackos, JoAnn T. (1994). *Managing Your Documentation Projects*. New York: John Wiley & Sons, Inc.

Howlett, Virginia (1996). *Visual Interface Design for Windows*. New York: John Wiley & Sons, Inc.

Lewis, Clayton and John Rieman (1993). *Task-Centered User Interface Design. ftp://cs.colorado.edu/put/cs/distribs/clewis/HCI-Design-Book.*

Mandel, Theo (1997). *The Elements of User Interface Design.* New York: John Wiley & Sons, Inc.

Microsoft Corporation (1993). *The Windows Interface Guidelines for Software Design.* Redmond, WA: Microsoft Press.

Microsoft Corporation (1995*) Microsoft Windows 95 Help Authoring Kit.* Redmond, WA: Microsoft Press.

Nielsen, Jakob (1997a). *How to Conduct a Heuristic Evaluation*. http://www.useit.com/papers/heuristic/heuristic_evaluation_html.

Nielsen, Jakob (1997b). *Characteristics of Usability Problems Found by Heuristic Evaluation.* http://www.useit.com/papers/heuristic/usability_problems.html.

Nielsen, Jakob and Darrell Sano(1997). *SunWeb: User Interface Design for Sun Microsystem's Internal Web*. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/HCI/nielsen/sunweb.html.

Nielsen, Jakob (1996) *Seductive User Interfaces*. http://www.useit.com/papers/seductiveui.html

Rubin, Jeffrey (1994). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests.* New York: John Wiley & Sons.

Rubin, Jeffrey (1994). "Conceptual Design: Cornerstone of Usability." *Technical Communication*, 2d Quarter, 130-138.

Schneiderman, Ben (1992). *Designing the User Interface*, 2d ed. Reading, MA: Addison-Wesley Publishing Co.

Spool, Jared (1996). "Is Your Application a Core or a Ring?" *Eye for Design*, November/December, 1-3.

Weinschenk, Susan, et al. (1997). *GUI Design Essentials*. New York: John Wiley & Sons, Inc.

Zetie, Carl (1995). Practical User Interface Design: Making GUIs Work. New York: McGraw-Hill, Inc.

University of California—Davis. *Application Development Methodology*. http://sysdev.ucdavis.edu/WEBADM.

Contact Information

John E. Bentley
First Union National Bank
400 S. Tryon Street, NC-0094
Charlotte, NC 28285
704-383-2686 or John.Bentley2@FirstUnion.Com