Paper 166

GUI KISSes: Tips and Strategies for Interface Design

John A. Quarantillo, Westat Inc., Rockville, MD

ABSTRACT

This paper discusses Graphical User Interface (GUI) design. It is tempting for developers to build an interface using too many ’bells and whistles’. Such an interface, although technically appealing and challenging, can be difficult to develop and maintain, but most important, difficult to use. The acronym KISS (Keep It Simple, Stupid) applies well to interface design. A simple, effective interface should be designed with the users’ needs taking first priority. Specific topics addressed include: which types of screen controls to use and when, keyboard and mouse support, designing with portability in mind, and testing. The audience level is beginner, and will focus more on the strategies used than the actual code to implement such interfaces.

INTRODUCTION

GUI Design has come a long way in the past few years. What is now possible and sometimes commonplace was previously unthinkable, or unaffordable. With the availability of integrated design environments, developers today can quickly and effectively produce simple graphical front ends to complex databases. The tips and strategies discussed here are the result of eight years of development using a variety of packages including SAS® software (SAS/FSP® FSEDIT procedure and SAS/AF® with and without FRAME entries), Microsoft Access®, and Borland (now Inprise) Delphi®. A variety of front and back end configurations were developed, based on client needs and which products were the best fit to the problem at hand and the staff available.

ASSIMILATE

A good systems analyst is able to take a problem and digest it, absorb the problem, learn about it, grasp it, and fully understand it. This is an often overlooked or minimized step in the development of an interface. Developers and analysts might be well versed in the system requirements from the systems point of view, but they must also look at things from the users’ perspective. Before the first screen is drawn, the developers need to understand fully the way in which the interface is to be used. It is too easy for developers to second-guess the users while meeting all the back end systems requirements. In most cases the guesswork falls short. Yes, there are many details about a system that the users aren’t aware of and need not be, but they are well versed in how they want and need to do things. If the designers aren’t aware of the full system requirements, from the front (user perspective) and the back (system perspective) they are certain to fail in designing an interface.

SIMULATE

When designing an interface, sometimes the best approach is to simulate the current environment. If this is a system that is to be used to collect data from hard copy, the best design for an input screen is one that most closely resembles the currently used paper form. If you are designing a billing system, make the screen look like the old invoices. If you are collecting employee job histories, make the screen look like the companies’ employment records. If you are collecting death certificate information, make the screen resemble the actual certificates. The benefits reaped by designing an interface in this manner are incredible. These benefits are so large, that if the current paper system has historically different layouts of the same hard copy data forms, it is worthwhile to develop multiple input screens to match each historical revision of the data form. In the previous death certificate example, if you are building a system to collect data from a number of different states it might be worthwhile to design a separate screen for each state that has a different format for their certificate.

If the system being developed is a replacement of a current computerized system, this approach is still valid, but to a point. Obviously the current system has shortcomings, because it is being replaced. In this case, the goal of the developers is to simulate the feel and appearance of the existing system, while enhancing its overall performance. By keeping the overall feel of the existing system, the training and transition periods will go much more smoothly.

HOMOGENIZE

Effective data systems have a similar feel throughout the entire system. The users find comfort and familiarity when the same basic menu options appear on all screens, and in the same space. Certainly some screens will have menu options unique to that screen, but the basic menu options should appear exactly the same and in the same order throughout the system. The following is a good example that will illustrate the need to homogenize.

> A large system was developed to collect medical information in a clinical trial study. The main portion of the system was designed, developed and implemented by a large team of programmers that communicated closely and set the standard that on all menu screens, the hotkey “x” would be used to exit that menu and move up one level in the system. The problem surfaced when a separate programmer was given the task of developing a data transmission module to add in to the system. This programmer was not aware of the standards and conventions set by the main team. In the data transmission portion of the system the hotkey “e” was used to exit that menu and move up one level in the system. This seems like a minor issue, but was compounded by the fact that on one of the menus the hotkey “x” was used to begin the “exchange data” portion of the system, which could take sometimes more than an hour to run.

Another point to consider is color. Color can be useful to alert the user to problem areas. Color can also be useful to indicate to the user which part of a system they are in. Careful consideration must be taken when using color. First, if the system is to be ported to another platform, the developers must ensure that the color scheme used is functional on the platform intended for use. Certain color schemes that offer bright contrast in a Microsoft Windows® environment might prove to be virtually unreadable on an MVS mainframe. Second in the Windows environment, if the developers use system default colors such as background, and buttonface, they might be quite surprised what their interface looks like when a creative user ventures into Control Panel and changes the color scheme to fit their taste. It is recommended that developers avoid using these system defaults. Choosing a good color scheme, and hard coding the interface to that scheme is the best way to prevent these problems.

This homogenization requires a bit of communication among the design and development teams. The best approach is to set standards early, make them known to all the design and development teams and ensure that all involved stick to these standards.

EMPOWER

An effective interface will give the user the ability to quickly and easily do what they want with the data. An important facet of interface design is to choose the proper screen controls to fit the type of data. GUI interfaces can include a vast array of screen controls that allow the user to enter and view information almost instantaneously. Certain types of data fields or columns lend themselves to specific screen controls. Generic terms are used here to describe screen controls, as each package has its own ‘special’ naming convention.

Simple free flow text input/update.	Text Box.
Read only text.	Label, or a disabled Text Box.
Simple questions i.e. retired? deceased ?	Checkbox type control. Check meaning yes, no check meaning no.
Mutually exclusive group of simple questions i.e. Shipping Method: USPS, UPS, FedEx)	Radio Group type control, where only one of the choices can be selected.
Lookup type variable i.e. State.	Combo Box type control. This control opens up to a list of choices, but closes to show the choice specified.
Browse for the answer. The best example here is the Open File Dialog Box.	List Box type control. This control is the same as an opened combo box. Used when the user will always need to see the list of choices.

There are many other types of controls, but the important thing to keep in mind is to use the control best suited to the data or task. Another point to consider when choosing controls is overall appearance of the screen. Screens should be organized in a logical, straightforward manner. Care should be taken to avoid screens that look cluttered.

Unfortunately (for developers) users are quite varied in their wants and needs. It is important to keep in mind these different types of users when designing an interface. Many users are quite savvy, and flit about the screen with a combination of hotkeys, mouse clicks, and keystrokes. Other users plod along never bothering to find or use any shortcuts.

Still others seem to suffer from Murophobia (the fear of mice). Users who are good typists usually prefer hotkeys and keystrokes to any mouse use. They contend that moving from the keypad to the mouse slows them down. Building an interface that supports most, if not all these types of users is highly important. Platform issues arise here, as mouse input is not always supported. A well-implemented interface will allow the user to perform the same action using any and all input devices. The best way to achieve this full empowerment for all types of users is to create a triple redundant command interface that builds in mouse commands, hotkeys, and visual buttons to perform all basic and important tasks.

SUPPORT

Even if you follow all the suggestions previously mentioned, your interface will still require some kind of ongoing support.

The first facet of support is documentation. If you've designed your system correctly, you need to relay the information to the users. The best method to do this is a users' guide. Many systems even go the extra mile by building in online help. Either way the users need to know what to expect from the system, and need to know how to use the system. If you don't tell them about the shortcuts they won't find them. If you don't tell them how to use certain controls, they may not input data correctly.

The next phase of support is the testing phase. Testing to most developers is the process of zipping through the application and declaring that it works. The problem with this is developers don't think or act like users. It is paramount that testing be performed by someone other than the person that developed the system. Optimally potential users or persons at the same level of expertise should perform the testing. It is not uncommon that after initial feedback from testing developers are forced to reconsider some assumptions made during the design process, and rework a significant portion of the system.

The last phase of support is maintenance. After a system has passed internal testing, and perhaps beta testing by consumers, the system is released into production. At this point the system has entered maintenance mode. This is an ongoing process of communication between the developers and users. When a problem arises, the developers evaluate it and decide if the problem is worthy of delivering an upgraded version of the interface.

CONCLUSION

By following the aforementioned strategies, developers can design and implement effective and useful interfaces that are simple to design, develop, use and maintain.

ACKNOWLEDGMENTS

SAS, SAS/AF, and SAS/FSP are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

For Further Information Contact:

John Quarantillo
Westat Inc.
1650 Research Blvd. Rm. TB322
Rockville, MD 20850

quaranj1@westat.com