# Implementing Human Interface Guidelines

David Allen

September 27, 2015

# 1 Introduction

Nowadays, the majority of programs requiring user input provide forms for entering information, have the ability to digitize clicks on the screen, or employ various other graphical methods for entering data. Such a facility is called a *Graphical User Interface* or *GUI* for short. To aid in designing a *GUI*, many *Human Interface Guidelines* or *HIG*s have been proposed. This talk is about implementing *GUI*s with *HIG*s in mind. It is presented at a conceptual level, *i.e.* no computer code is shown. We will run some demonstration programs.

# Organization

This talk has three parts:

1. Selecting a programming language.
2. Selecting a *GUI* toolkit.
3. *HIG*s, or the philosophy of good *GUI* design.

# 2 Selecting a Programming Language

My interest is in efficiency. *C* and *C++* are the most popular compiled languages. They produce more efficient code than interpreted languages like *R* and Python. Toolkit developers tend to prefer *C++* over *C* because the graphical elements can be written as classes. Application developers can then use inheritance to modify the appearance of the elements and define the actions associated with an element.

# What is *C++*?

Scott Meyers [1] describes *C++* as a federation of languages. Here is a abbreviated description of the sub-languages:

- A better *C*. A *C* program with the exclusion of some dangerous constructs is a *C++* program.
- *C* with classes. This adds object oriented features including inheritance, polymorphism, virtual functions, *etc*.
- Template *C++* is the generic programming part of *C++*.
- The standard template library provides many containers and algorithms.

# 3 Selecting a *GUI* Toolkit

You don't want to develop a *GUI* toolkit from scratch, but instead select one of many that are available. Your choice will depend on your priorities. My priorities are

1. The toolkit must portable, *i.e.* it must run on the major operating systems: Windows, OS X, and Linux.

2. The toolkit must be free to use, and there should be no restrictions on software produced using the toolkit.

3. Various other things like ease of use, appearance, support, *etc.*

# Available Toolkits

See Wikipedia for a list of toolkits with some discussion of advantages and disadvantages of each. The following are the ones I gave at least a cursory examination.

- *Qt*, https://www.qt.io/
- *GTK+* (written in *C*), http://www.gtk.org/
- *wxWidgets*, http://wxwidgets.org/
- Fast light toolkit or *FLTK*, http://www.fltk.org

# Some Advice

I had programmed in *C* for several years when I decided I wanted to program a *GUI*. I tried to learn *C++* at the same time as I was evaluating toolkits. Don't do that. When there was a problem, I didn't know if it was a *C++* issue or a toolkit issue. Get a fair grounding in *C++* before attempting to use a *GUI* toolkit.

# *FLTK*

My selection was *FLTK*, but the selection process was not systematic. It could be that by the time I looked at *FLTK*, I had gained enough knowledge of *C++* that things just went better.

*FLTK* satisfies completely my first two priorities listed on page 6. My big surprise was support. There is an active discussion group that includes the developers and maintainers. They are patient and tactful when dealing with stupid questions. I have received responses to questions on the day of the posting.

# Programming with *FLTK*

A *widget* is a term to represent any element of a *GUI*. Examples include buttons, choice boxes, input fields, and text displays. *FLTK* provides functions for many different widgets. The programmer can tailor the widgets to their own purposes using inheritance. Some widgets, a button for example, require the programmer to supply a callback function that is executed when the widget is clicked.

*FLTK* is written using only the "*C with classes*" sub-language of *C++* (refer to page 5). However, users of *FLTK* are free to use all aspects of *C++*.

# Knocks on *FLTK*

In critiques of *GUI* toolkits, *FLTK* has received some knocks:

- The visual aspects of *FLTK* are plain.
- The callback function interface is primitive.

These defects (or features) are probably the reason *FLTK* is fast and light.

The visual aspects are plain. But to me, it is more important that good *HIG* principles are followed, than is is to be flashy. A primitive interface may make the interface easier to understand and provide more flexibility.

# 4 Human Interface Guidelines

*Human interface guidelines* are a set of recommendations for software application developers. Their aim is to improve the experience for the users by making application interfaces more intuitive, easier to learn, and consistent. Wikipedia provides a good discussion with many links.

# *FLTK* Guidelines

*FLTK* has posted its guidelines at

http://www.fltk.org/hig.php.

Their top ten points are

1. Know thy user, and YOU are not thy user.
2. Things that look the same should act the same.
3. Everyone makes mistakes, so every mistake should be fixable.
4. The information for the decision needs to be there when the decision is needed.

5. Error messages should actually mean something to the user, and tell the user how to fix the problem.
6. Every action should have a reaction.
7. Don’t overload the user’s buffers.
8. Consistency, consistency, consistency.
9. Minimize the need for a mighty memory.
10. Keep it simple.

# A Pet Peeve

The Opera browser introduced the *tabbed interface* for switching between pages and deleting pages. A nice feature is that when your delete the last tab, a *speed dial* of your favorite sites appears. You can then click an item on the speed dial, search, or quit.

Most other browsers and text editors have now adopted the tabbed interface, but have implemented it in different ways. In Firefox, if there is one tab remaining, and you delete it, the program terminates.

# A Test

I propose a test. When deciding between two actions to be associated with clicking a widget, write out the two actions and then assess relative to the guidelines.

In the browser example, suppose a page is deleted by clicking an × on its tab. Consider two actions

1. Clicking the × on its tab deletes the page.
2. Clicking the × on its tab deletes the page. If the page is the only page, the program also terminates.

# A Statistical Example

Most regression programs put an intercept term in a model by default. The user has to remove it if he doesn’t want it. Consider these instructions:

1. Specify the terms in the model.

2. The intercept term is included in the model by default — remove it if not appropriate. Specify the additional terms in the model.

# References

[1] Scott Meyers. *Effective C++*. Addison-Wesley, third edition, 2005.