Chapter 9: Human–Computer Interface Design

Jacob Somervell,
University of Virginia, College at Wise

CHAPTER OBJECTIVES

- Understand the role that the computer interface plays in high-quality and successful software systems
- Describe how to address interface design and evaluation within the software development life cycle
- Provide usable guidance for evaluating designs

CONCEPTUAL OVERVIEW

To end users, the interface is the system. Most end users do not know, nor do they even need to know, about the underlying structure and implementation of the software system. They are concerned only with the interface presented to them and the capabilities provided by that interface. Consider the analogy of driving a car: the driver does not need to know anything about how an internal combustion engine works and how it is connected to the transmission to send power to the wheels, nor do they need to know about hydraulics and fluid dynamics of the braking system to actually drive a car. Indeed, mainly the driver needs to know that "D" means drive (assuming an automatic transmission), press gas pedal to go, press brake pedal to stop. The same concept applies to software systems. End users are concerned only with what they can do with the system and how they do it, not with how it works "under the hood." Hence, it is of vital importance to get the interface design sufficiently correct so that it serves users in an efficient and usable manner. This chapter shifts the focus away from the detailed under-the-hood approach to design to cover the essential human–computer design activity. The chapter focuses on providing valuable information on how to create effective and usable interfaces for software systems.

WHAT IS HUMAN–COMPUTER INTERFACE DESIGN?

So what does it mean to "get it right?" As part of the software design process, human–computer interaction (HCI) design must account for the user of the software. While designing the architecture and detailed design of software systems is essential for meeting most quality attributes, designing an efficient user interface that is understandable by the end user is paramount to the usability quality of all successful software systems. The most elegant, efficient, and high-quality architectural and detailed designs can be felled by a poor interface. In the context of HCI, interface design refers to the creation of the user interface. IEEE (1990, p. 80) defines the user interface as follows:

An interface that enables information to be passed between a human user and hardware or software components of a computer systems.

For most software, this entails designing the graphical user interface (GUI). This involves selecting appropriate information presentation and interaction techniques for the various end-user classes (Rosson and Carroll 2002). More specifically this entails selecting appropriate information layouts, correct language, appropriate interface controls (e.g., radio buttons versus check boxes), and tying the detailed design to the various input mechanisms provided in the interface.

An essential task of the HCI design activity involves making sure that the interface provides appropriate means for using the system in an efficient manner. The best way to ensure an interface is sufficiently good is to iteratively improve the design through user testing. The implication is that there will be multiple iterations of a process that includes the following HCI design tasks:

- Creating a prototype of the system
- Having end users use that prototype in realistic ways
- Gathering data from these tests
- Redesigning the interface to address discovered problems

All of this work (choosing appropriate information representation and interaction methods) hinges on a thorough understanding of the users of the system. It is paramount to learn how the users typically perform similar actions and what their expectations of the new system may be. This information is gleaned through detailed requirements gathering and analysis and through significant user testing.

WHY STUDY HUMAN–COMPUTER INTERFACE DESIGN?

The HCI design activity is where general principles are applied to optimize the interface between humans and computers. Visual designs have a major role in the success or failure of software systems. Systems that meet functional requirements but are not usable cannot succeed. The major concerns of the HCI designs may include the evaluation and use of modes, navigation, visual designs, response time and feedback, and design modalities, such as forms and menu-driven. HCI designs directly influence the quality of any system and are essential to understanding and addressing the factors that affect the overall usability of the system. Many design principles and evaluation techniques exist to successfully design user interfaces. Therefore, understanding the techniques and tools for designing interfaces allows designers to become proficient in creating efficient interfaces. Providing an interface that allow users to accomplish their goals with the software, without unnecessary effort, is the ultimate goal of the user interface designer.

The "without unnecessary effort" clause is important. Consider a system that requires a date from the user, as presented on Iteration 1 of Figure 9.1.

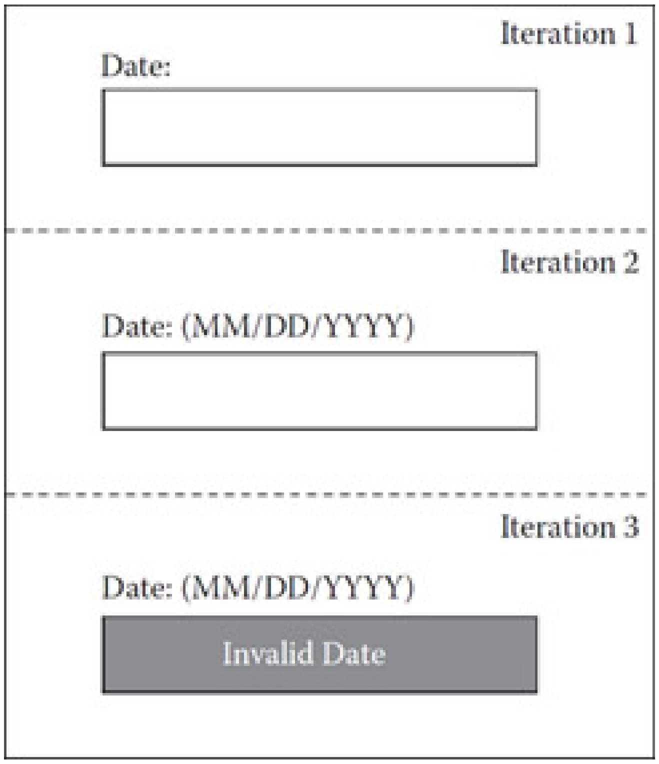


Figure 9.1: Simple data entry user interface

The user interface could ask the user to type in the date in a text box. Without any extra information a user could type any of the following:

- Jan 1, 2011
- 1/1/11
- 1 −1 −11
- 1 January 2011

Which format was expected by the software? If the information typed by the user doesn't match the format, what happens? An error message? Program crash? The most flexible option involves allowing all of these and others as valid input and then correctly parsing the input to find the appropriate fields for month, day, and year from that input string. This is a nontrivial solution and does not help the user form a basis or understanding of the desired input. A simpler solution involves adding a label to the input box specifying the appropriate format (i.e., MM/DD/YYYY), as presented on Iteration 2 of Figure 9.1. This option works fine and helps the user understand the expected formats for date objects, knowledge that can be leveraged in other software systems with similar requirements. Finally, to address input errors, error indicators in the interface are introduced to help guide the user to the correct input format, as presented on Iteration 3 of Figure 9.1. Typically, a red error indicator (e.g., as a background color on input boxes) is a passive way of accomplishing this.

Many other options for date input could be used instead, including drop-down lists, graphical mini-calendars for selecting the date, and spinners. Each of these options has been used with varying levels of success in various applications. Learning which method works for a specific software system requires end-user testing: create prototype systems or mock-ups with each design choice, have users complete realistic tasks using these systems, record information about the users' performance (e.g., speed, accuracy), and empirically determine the best design. This effort is significant and requires time and resources to accomplish correctly.

The point of this discussion is that the interface (the presented information to the user) plays a significant role in the utility of the software and the experience of the user. Carefully considering the user and the user's abilities when designing the interface can only increase the usability of the software system. Thorough testing of the design, with end users, is paramount to designing successful software systems.

So how do we go about involving the user in the process? One must spend time early in the software development life cycle identifying the user classes for the software. A user class is a set of users who share common tasks with the software. Consider a student information system that may be used by a university. Clearly, one class of user would be the students. Another obvious class would be the faculty. Other classes that might not be immediately obvious could include department chairs, registrar, advisors, enrollment management staff, and financial aid staff. There could be others. Furthermore, a single person could take on the characteristics of multiple classes. For example, a faculty member could also be an advisor and a department chair. Each of the user classes to which a user belongs contributes to the type of work that that user should be able to perform with the software.

After identifying the user classes, it is then necessary to hold requirements meetings with representatives of each user class. These meetings should elicit the tasks that the target user class should be able to do with the software as well as tasks that would be "nice" to have. Recall that these meetings would normally occur as part of the requirements gathering process in the overall software development life cycle, not as a separate activity, although it could be separate if needed. Similar to software architects, it is common for user interface designers to go back and forth between requirements and design, until the interface is sufficiently appropriate to accommodate the needs of users of each class.

Skill Development 9.1: Eliciting Needs from Different User Classes

Create a list of 10 questions you would ask a group of students about their expectations for a new student information system. Create a list of 10 questions you would ask a group of faculty about their expectations for the same system.

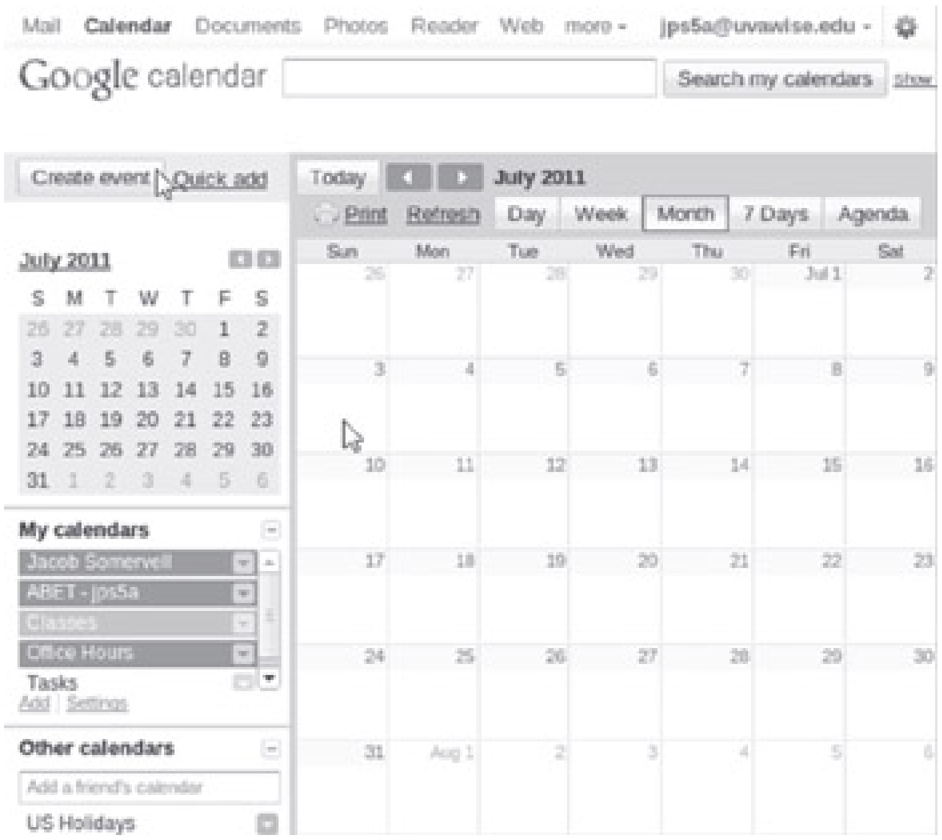
After learning about the typical interactions your user classes will have with the system, one can begin designing the user interface to meet those needs. One of the most challenging things for software engineers is to disconnect from the system they are designing and try to see the world through the eyes of the end user. It is tempting to simply design an interface that meets the needs of the engineer or developer (e.g., for testing, verification) instead of designing an interface that meets the needs of the end user. It is up to software developers to maintain a clear focus on the end user while developing the user interface for a system. The following sections provide specific information to aid the designer in focusing on the user.

GENERAL HCI DESIGN PRINCIPLES

Several design principles and heuristics exist for guiding user interface designers. According to Nielsen and Mack (1994), there are 10 major heuristics to follow when creating a user interface:

1. *Visibility of system status:* The interface should have some mechanism for showing where users are in their task.
2. *Match between system and the real world:* The interface should provide interaction techniques that mimic or model what is expected in the real world.
3. *User control and freedom:* The interface should support user exploration without fear of breaking anything. Undo and redo should be supported.
4. *Consistency and standards:* The interface should use, for example, language or wording that is consistent with users' expectations. Follow style guides and platform standards.
5. *Error prevention:* The interface should help users avoid mistakes. Always ask them when they initiate a destructive command.
6. *Recognition rather than recall:* The interface should support rapid and easy learning of the system and support recognizing features and their associated actions rather than relying on memorization of unique interface widgets.
7. *Flexibility and efficiency of use:* The interface should provide users with shortcuts or other accelerators. This helps the interface get out of the way of expert users while allowing novice users the opportunity to become more efficient.
8. *Aesthetic and minimalist design:* The interface should present only the necessary information and no more. Extra visual elements can distract from the important information.
9. *Help users recognize, diagnose, and recover from errors:* Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. *Help and documentation:* Make sure the help and documentation is clearly available in the interface.

These 10 guidelines or heuristics are generic and open. This is intentional so that they can be applied across a large cross section of software systems. This generality can sometimes lead to ambiguity and confusion on the part of the developer. More specific, heuristics can be useful when developing software for specific platforms or systems (Somervell and McCrickard 2005). However, the point is that there are some simple, straightforward things to consider when designing any user interface. More importantly, these 10 guidelines are focused on users and strive to keep them in control of the system: the system serves the user. Applying these rules in a specific application can help create an interface that users will find both useful and usable. Consider a popular online calendar application as presented in Figure 9.2.



In this particular calendar application, creating an appointment relies mostly on direct manipulation—directly clicking on the desired day, typing in a description, including time, and hitting the return key. This action is analogous to writing that information on a desk calendar or other paper calendar. It is exactly what users expect to be able to do with a calendar. In addition, there is a button ("Create event") that allows users to add appointments by filling in a form: some users may prefer this method of entry, especially for events that occur in the future, which would require navigation within the calendar to enter through the direct manipulation route. A new appointment is shown immediately in the day for which it is assigned.

Now consider the 10 guidelines in relation to this interface. In terms of system status this interface shows an entire month (or week or day) of appointments in the expected monthly format. A newly created appointment appears immediately within the day for which it occurs. Error conditions are displayed prominently in the top center of the display in red text. In terms of matching the real world, it clearly mimics the desktop calendar and heavily utilizes that metaphor. The form and layout closely resembles other calendar applications so consistency and standards are followed. Users can really commit no errors in the software, and any mistakes are easily corrected through undo or deletion capabilities. As a GUI application, there are no commands to learn and remember from usage to usage, and help is available through pop-up labels on hover. The design is minimal in that only the necessary information is shown with extra functionality available in submenus. Overall, this particular interface performs well with respect to Nielsen and Mack's (1994) heuristics.

This high-level analysis of the calendar application serves to illustrate a good interface and how a good interface will typically meet most of the general heuristics. Improvements to an interface will often revolve around one of the 10 areas described by these guidelines. However, these guidelines are just that: guidance. How does one go about starting the interface design process? How do we get a prototype of the system?