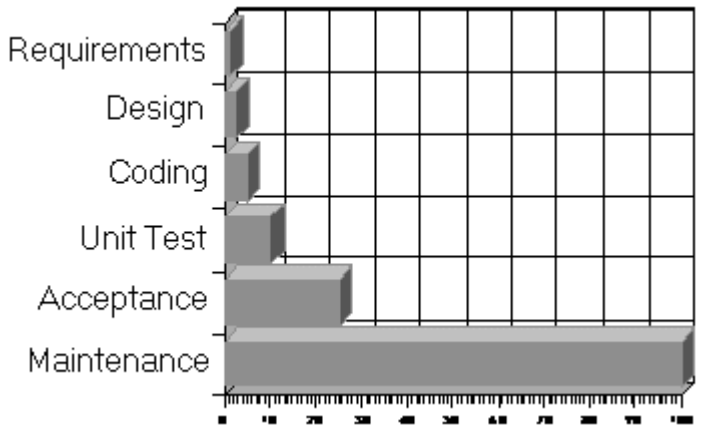
Why are Accurate Requirements so Important?

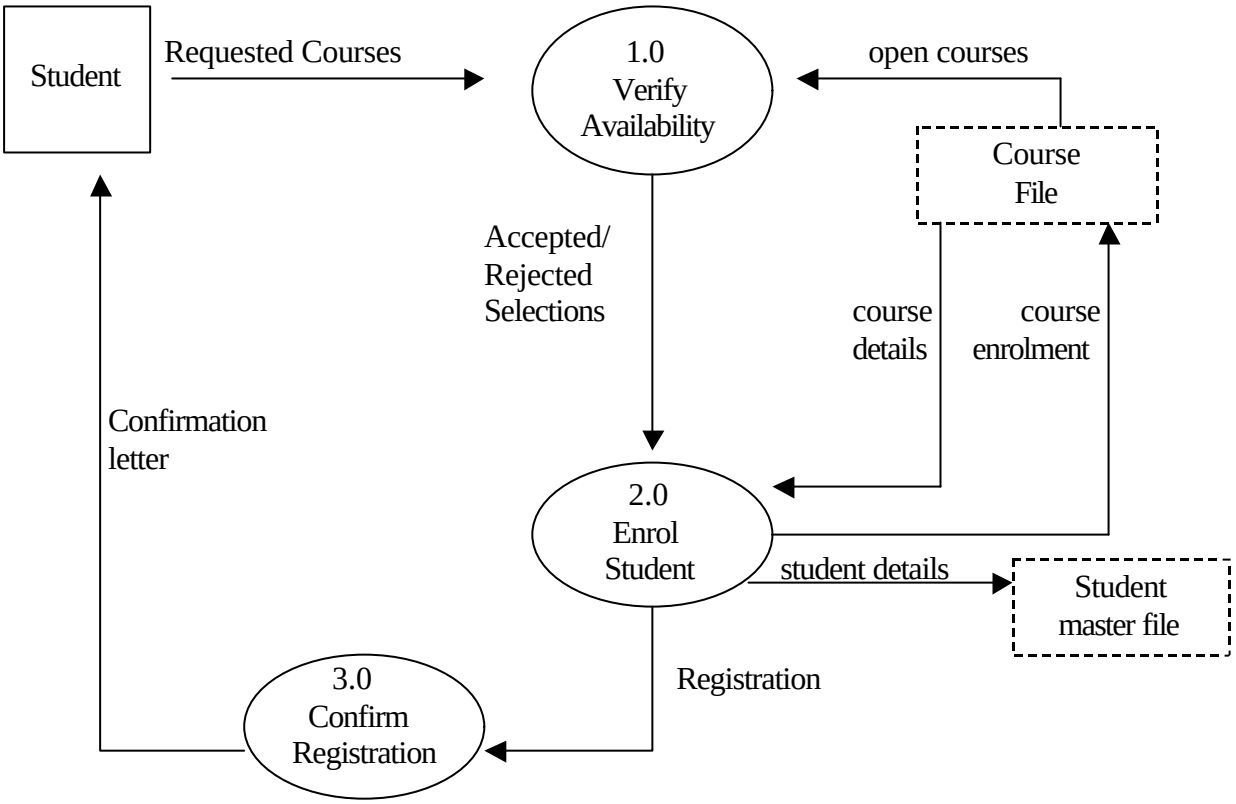
- **Inaccuracies Cripple Projects**
  The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.
  Fred Brooks - "No Silver Bullet"
  One of the most frequent causes of failure in software projects is a non-existent or inadequate statement of what the software is supposed to do. A relatively small amount of effort spent in accurately specifying software requirements has repeatedly generated large returns in developer productivity and customer satisfaction.
- **Leverage Limited Resources**
  Improving your organisation's competence in requirements capture will apply your limited resources at the point of greatest leverage to increase customer satisfaction and development team productivity. Research has shown that although requirements analysis consumes merely 5 percent of software production cost it affords 50 percent of the leverage to influence improved quality and productivity.
- **Poor Requirements Waste Money**
  A focus on quality requirements cuts development costs. It takes 100 times more effort to correct errors discovered in the maintenance phase than it does to correct the same errors if they are discovered in the requirements phase of a project.
- **Relative cost to repair bugs resulting from defective requirements by project phase.**



- **Build Quality In**
  Correct requirements build quality into your product from the start. Attempting to test quality into code at the end of a project is expensive and ineffective. The probability that defects originating from defective requirements will be completely rectified decreases as they are allowed to propagate into designs and code.

Data Flow Diagrams

A DFD is a graphical representation of a systems component processes and the interfaces (flow of data) between them.



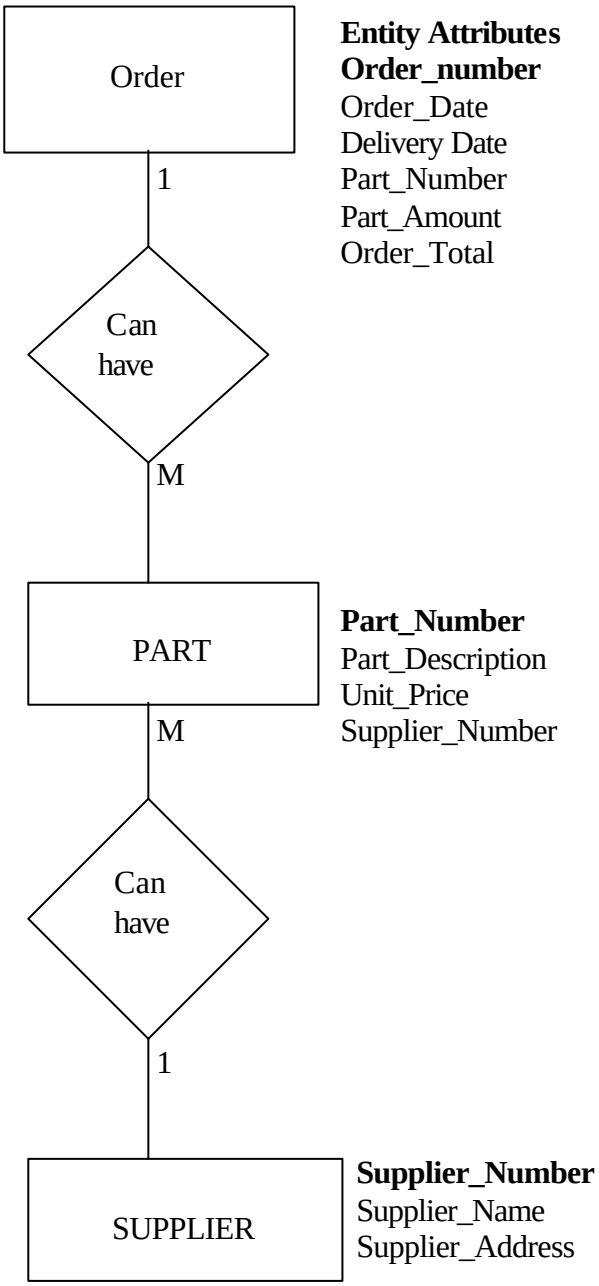
Above is a simple data flow diagram for a mail in university course registration system. The rounded boxes represent processes, which portray the transformation of data. The square box represents an external identity, which is an originator or receiver of information located outside the boundaries of the system being modelled. The open rectangles represent data stores, which are either manual or automated inventories of data. The arrows represent data flow, which show the movement between processes, external identities and data stores. They always contain packets of data with the name or content of each data flow listed beside the arrow.

This data flow diagram shows that students submit registration forms with their name, identification number and the numbers of the courses that they wish to take. In process 1.0 the system verifies that each course selected is still open by referencing the university's course file. The file distinguishes courses that are open from those that are full or cancelled. Process 1.0 then determines which of the students selections can be accepted or rejected. Process 2.0 enrols the student in the courses for which they have been accepted. It updates the university course file with the student's name and id number and recalculates the class size. If maximum enrolment has been reached the course number is flagged as closed. Process 2.0 also updates the university student master file with information about new students or changes in address. Process 3.0 then sends each student applicant a confirmation of registration letter listing the courses for which they have been registered and noting the course selections that cannot be fulfilled.

The diagrams can be used to depict higher-level processes as well as lower level details. Through levelled data flow diagrams a complex process can be broken down into successive levels of detail. An entire system can be divided into subsystems with a high-level data flow diagram. Each subsystem in turn can be divided into additional subsystems with second level data flow diagrams, and the lower level subsystems can be broken down again until the lowest level of detail has been reached.

Entity Relationship diagram

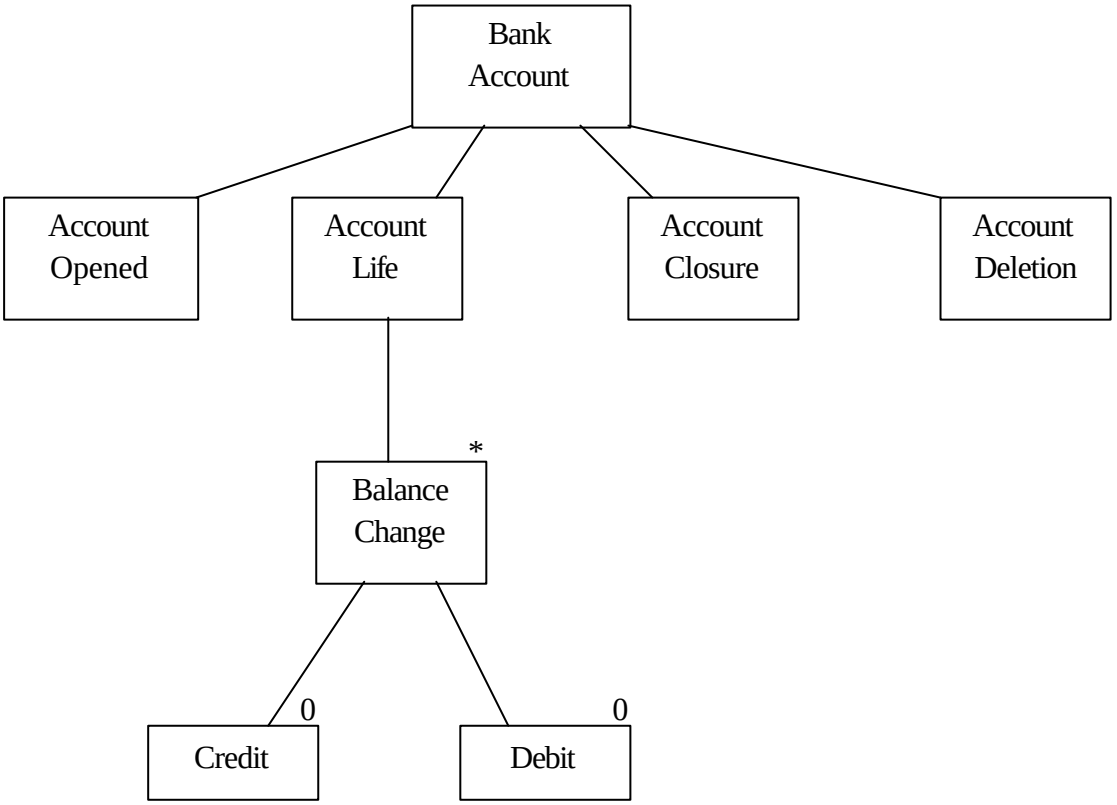
Database designers document the conceptual data model with an entity relationship diagram.



The boxes represent the entities and the diamonds represent the relationships. The 1 and the M on either side of the diamond represent the relationship among entities as either 1 to 1, 1 to many, or many to many. The entity ORDER can have more than 1 PART and a PART can only have 1 SUPPLIER. Many parts can be provided by the supplier. The attributes for each entity are listed next to the entity and the key fields are in bold.

Entity life History

Entity life histories model the system from the viewpoint of how information is changed. What the entity life histories show is the full set of all information changes that can possibly occur within the system together with the context of each change.



An entity life history is a diagrammatic representation of the life of a single entity from its creation to its deletion. The life is expressed as the permitted sequence of events that can cause the entity to change. An event may be thought of as whatever brings a process into action to change entities, so although it is a process that changes the entity, it is the event that is the cause of the change.

- Sequence
  The boxes read left to right from account opened to account deleted. This is the only possible sequence and there is no indication of the time between the boxes in a sequence.
- Selection

The boxes with a 0 over the top rh corner are alternatives for one another; each balance change is either a debit or a credit. Nodes that are alternatives to each other.

- Iteration

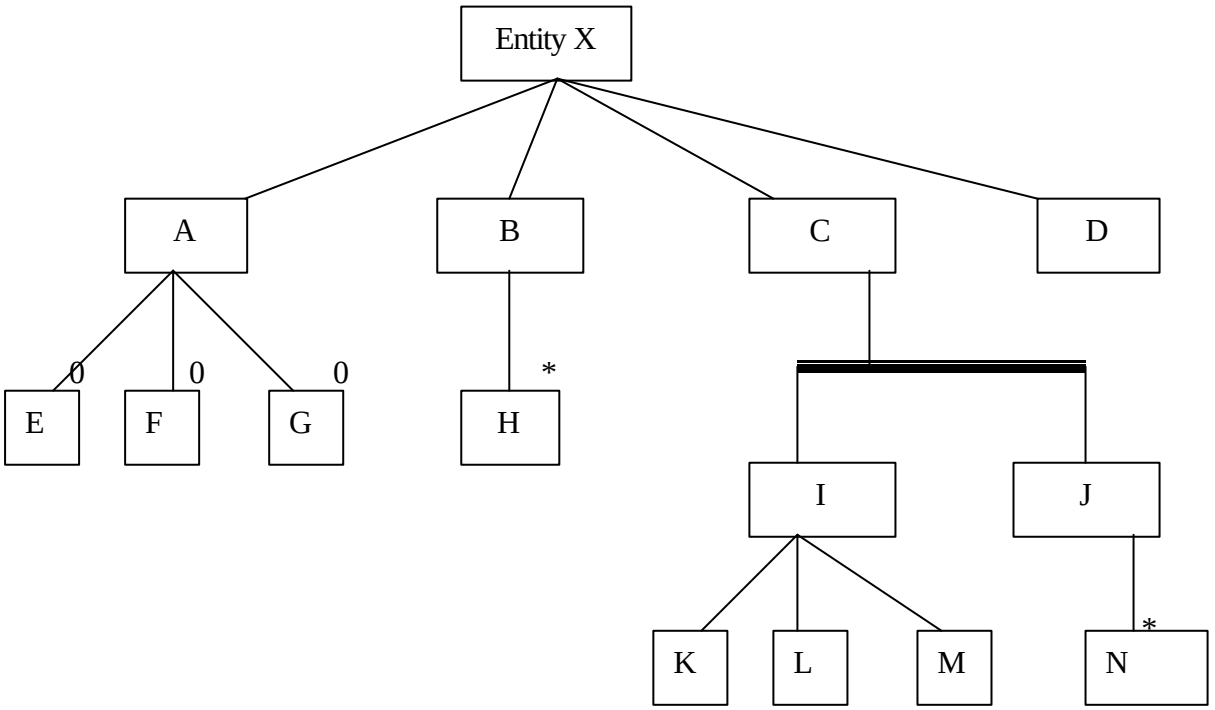
  The box with the * over the rh corner represents an iteration, many balance changes can occur one after another. A node that may be repeated many times over.

Diagrams using these 3 components are called "Jackson structures" (or Jackson diagrams) after Michael Jackson who pioneered their use as a technique for program design in the early 70s.

All entity life histories can be built using just these 3 components. Certain complex situations can be simplifies using 2 other conventions

- Parallel structures
- Quit and resume.

A parallel structure is used where nodes occur in no predictable sequence



This represents the situation where the sequence of K, L and M occurs at this point in the Entity life history and the event N may affect the entity a number of times during this sequence.

The quit and resume convention is used in situations where the diagramming conventions excessively constrain the entity life history or force a very complex artificial structure in an attempt to model a particular situation. The use of this convention allows a quit from one part of an entity life history diagram to resume in another part of the diagram.

The representation of user requirements is a crucial aspect of the systems analysis process. The quality of implementation models used to build a system, and of the ultimate system itself, are dependent largely on the extent to which systems analysis models faithfully represent users' requirements (Jarvenpaa and Machesky, 1989). Further, the need for users to validate requirements documents necessitates systems analysis models that communicate requirements clearly and effectively (Larsen and Naumann, 1992). Modeling devices may be more or less formal, and the degree of formality has implications for user understanding (Fraser, Kumar, and Vaishnavi, 1991, 1994). However, there have been no empirical tests of the degree of formality that is optimal in systems analysis models. The present research seeks to help fill this gap by providing an empirical test of various representational devices.

A variety of methods have been developed to help structure the systems analysis and design process and to communicate system requirements. The methods vary in terms of their degree of formality. For example, Dart et al. (1987) classified methods as "informal," "semiformal," or "formal" (see also Fraser et al., 1994). Informal methods are those that do not have complete sets of rules that regulate the types of representations that can be created. Examples of these methods include natural language text and informal pictures or diagrams. Semiformal methods have specified rules for creating the representations. Examples include data-flow diagrams (DFDs) and entity-relationship diagrams (ERDs). Formal methods are those with a mathematical foundation and rigorously specified syntax. Examples include petri nets and executable specifications (see also Denning, 1991).

Semiformal diagrams can be used to develop implementation-oriented models that guide system builders in the construction of applications. However, these diagrams may be difficult to use in the communication process between analysts and users (Lohse, Min, and Olson, 1995). For example, Fraser et al. (1994) state that "The very formality which makes formal specifications desirable during the later phases of requirements specification makes them an inappropriate tool for communicating with the end user during the earlier requirements elicitation and confirmation stages" (p. 75). Larsen and Naumann (1992) note that easily understood and verifiable representations are not useful for system construction, and representations with enough detail, precision, and rigor for building systems are not likely to be understandable to users. Despite these concerns, the predominant methods recommended by systems analysis and design texts for representing user requirements are semiformal methods (e.g., Whitten, Bentley, and Barlow, 1994). However, techniques used in practice often include informal methods (Fraser et al., 1991; Fuggetta et al., 1993), despite the fact that there is little guidance for the analyst in the use of such methods.

Users will understand information requirements represented using informal representational devices better than they will requirements represented using semiformal representational devices.

What is a "User Interface?"

The term "User Interface" refers to the methods and devices that are used to accommodate interaction between machines and the human beings who use them (users). User interfaces can take on many forms, but always accomplish two fundamental tasks: communicating information from the machine to the user, and

communicating information from the user to the machine. Any machine that requires interaction with human beings will have some sort of user interface. The devices that are used to implement user interfaces on modern computers are video screens, keyboards, and pointing devices such as mice and track balls.

The Evolution of User Interfaces

The very first computers had user interfaces that were as rudimentary as the computers themselves. The computer communicated information to the user through flashing lights, and the user communicated information to the computer by setting mechanical switches. Only highly trained specialists were able to actually communicate directly with a computer.

The next stage of evolution had computers communicating to users through printing devices, and the users communicating to the computer through punch cards. This was an improvement, but still cumbersome and inefficient. It was still rare that anyone but a computer specialist would actually communicate directly with a computer.

User interfaces left the dark ages when video screens were used to communicate information from the computer to the user, and typewriter-style keyboards were used to communicate information directly to the computer. This major innovation helped to allow "ordinary" users to communicate directly with a computer. But since the video screens were limited to displaying only the characters that were found on the keyboard, the usefulness of the user interface was constrained by the same limitation. Users were required to memorize commands that were generally tailored more to the computers than the users. A great deal of training was required before anyone could make use of the computer.

User interfaces entered the modern era when innovative designers at the Xerox Palo Alto Research Center broke away from the character-based interface paradigm and invented the Graphic User Interface (GUI). There were two major factors that separated the new paradigm from the old. One was the use of graphics to communicate information to the users visually in addition to textually. The other was to present a finite number of options to the users rather than requiring the users to memorize and manually enter commands from a virtually unlimited set of options. In this way the interface was focused on the needs of the human beings, rather than the other way around. This significantly reduced the training that was necessary to use a computer, and for the first time uninitiated users were able to become productive almost immediately.

The advent of these intuitive and easy to use design elements does not, however, ensure that interfaces which incorporate them will be intuitive and easy to use. As much as anything else, the advent of the graphic user interface has served to heighten awareness of general design principles that apply irrespective of the paradigm in which a user interface is implemented. This document discusses those principles as well as the GUI design elements and the best ways in which to use them.

"Quality" User Interfaces

The objective of any user interface developer should be to design and implement quality user interfaces. It's not always that easy to define what is meant by a "quality user interface." A "quality user interface" shall be defined as any user interface that is

intuitive, easy to use, and allows the users to maximize their efficiency and effectiveness when using it.

The best way to ensure quality user interface design is to use an orderly and well defined design process that is specifically geared to producing quality results. This applies to the design of the application that the user interface supports as well as the design of the user interface itself. The best user interface in the world will not be well received if the application itself is poorly designed.

An Overview of the Design Process

Below are some typical phases.

Phase	Description
Requirements	Determine the requirements for the application
Conceptual Design	Model the underlying business that the application will support
Logical Design	Design in general terms how the application will operate
Physical Design	Design in specific terms how the application will be constructed
Construction	Construct the application
Usability Testing	Test the usability of the user interface

The Requirements Phase

If the application is to be accepted by those who have a stake in it (stakeholders), it is imperative that they be involved from the very beginning. They should be sought out and polled as to what they consider their requirements for the application to be. The more broad a representative group of stakeholders is involved, the more broad the acceptance will be when it is delivered.

Below are some steps that will help lead to a successful requirements phase.

- Assemble the design team
    - Identify all stakeholder groups.
    - Select representatives to participate on the design team.
- Gather Requirements:
    - Interview as many stakeholders as is practical to determine:
        - What the underlying business problems are that this application should address
        - What benefits the application should provide
        - What the critical success factors are
- Define the scope of the project
    - Review the requirements that have been gathered
    - Make decisions about what will be included and what will not
    - If the scope of the project gets too big, consider breaking it down into stages.

Any user interface, no matter how well designed, won't be well received if its users feel disenfranchised from the design process.

The Conceptual Design Phase

This phase is concerned with modeling the underlying business that the application will support. User interface considerations are not addressed at this time.

Logical Design

The Logical Design phase gets into more specifics about how the application will support the business that was modeled in the Conceptual Design phase. The prototyping of user interfaces should kick off the Logical Design phase. By determining what events will occur on the client (e.g. clicking a button or selecting something from a scrolling list), the logical processes that support them can be designed. These events and processes are then broken down into "system functions."

The requirements about what the user interface should be able to do can drive the decision of what technology ultimately gets selected.

Along these lines, it is important to determine the minimum hardware configuration the application will cater to. For example, a particular user interface might look very good on a large color monitor, but be very difficult to use on a small black and white monitor. The decision must be made as to whether the user interface will cater to the small black and white monitor, or if it's okay to design around the large color monitor. Both solutions are valid, depending on the circumstances, but it must be a conscious decision.

Physical Design

This phase is concerned with determining how the logical design will be implemented on specific physical platforms.

This phase will not directly impact user interface design, unless it is revealed that it is not possible to physically implement certain aspects of the logical design.

Construction

When the users get something in their hands that actually functions, they will inevitably change their minds as to how they want things to work.

For this reason it's important to get functional interfaces in the users' hands as early as possible. If they request a change that winds up being somewhat fundamental, less re-designing will be required the earlier the change is identified.

Usability Testing

Usability Testing is a technique that can validate the user interface design and reveal areas in which it requires refinement. The basic concept is to simply observe users as they operate the interface. They should be instructed to verbalize their thought process as they go along. For example, they should be saying things like, "I want to find an invoice for a customer. I see a button that says, 'Invoices,' but I don't know if that will display one or create a new one..." By understanding what the testers are thinking it

will be possible to ascertain where they are having problems understanding and using the user interface.

It should be noted that this is often more an exercise in testing how easy a user interface is to learn than how easy it is to use. If the learnability of the user interface is the primary goal of the design, uninitiated users should be selected, they should be given minimal instruction or guidance, and the observers should look for areas in which the testers are having trouble figuring the user interface out. If the ease of use of the interface is the primary goal, novice users should be selected and the observers should look for areas in which the testers are having trouble generally using the interface or remembering how certain things work.

Either way it is very important that this technique not be used as a substitute for fundamental interface design. Usability testing is a process of validation and refinement. It is not part of the design process itself.

The more testers that participate in this exercise, the better the results will be. If one or two users have trouble in one particular area, that might not necessarily indicate a problem. But if a majority of testers run into the same problem or similar patterns emerge, it can be apparent that certain parts of the user interface require attention.

Concepts of User Interface Design

Learnability vs. Usability

Many people consider the primary criterion for a good user interface to be the degree to which it is easy to learn. This is indeed a laudable quality of any user interface, but it is not necessarily the most important.

The goal of the user interface should be foremost in the design process. Consider the example of a visitor information system located on a kiosk. In this case it makes perfect sense that the primary goal for the interface designers should be ease of operation for the first-time user. The more the interface walks the user through the system step by step, the more successful the interface would be.

In contrast, consider a data entry system used daily by an office of heads-down operators. Here the primary goal should be that the operators can input as much information as possible as efficiently as possible. Once the users have learned how to use the interface, anything intended to make first-time use easier will only get in the way.

User interface design is not a "one size fits all" process. Every system has its own considerations and accompanying design goals. The Requirements Phase is designed to elicit from the design team the kind of information that should make these goals clear.

Metaphors and Idioms

The True Role of Metaphors in the GUI

When the GUI first entered the market, it was heralded most of all for its use of metaphors. Careful consideration of what really made the GUI successful, however, would appear to indicate that the use of metaphors was actually a little further down in the list. Metaphors were really nothing new. The term computer "file" was chosen as a metaphor for a collection of separate but related items held in a single container. This term dates back to the very early days of computers.

The single most significant aspect of the GUI was the way in which it presented all possible options to the users rather than requiring them to memorize commands and enter them without error. This has nothing to do with metaphor and everything to do with focusing the user interface on the needs of the user rather than mandating that the user conform to the needs of the computer. The visual aspect of the GUI was also a tremendous advancement. People often confuse this visual presentation with pure metaphor, but closer inspection reveals that this is not necessarily the case. The "desktop" metaphor was the first thing to hit users of the GUI.

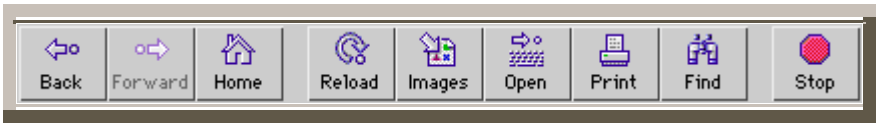
Metaphors vs Idioms

Most visual elements of the GUI are better thought of as idioms. A scroll bar, for example, is not a metaphor for anything in the physical world. It is an entirely new construct, yet it performs an obvious function, its operation is easily mastered, and users easily remember how it works. It is the visual aspect of the scroll bar that allow it to be learned so quickly.

Metaphors Can Hinder As Well As Help

The use of icons as metaphors for functions is a good example. It can be a gamble if someone will understand the connection between an icon and the function.

Consider the Microsoft Word 5.0 toolbar. Some icons area readily identifiable, some are not. The unidentifiable icons can be utterly perplexing, and rather than helping they can create confusion and frustration. And with so many pictographs crammed into such a small space, the whole thing reads like a row of enigmatic, ancient Egyptian hieroglyphs.



The Netscape toolbar, by contrast, can be considered to be much more graceful and useful. The buttons are a bit larger, which makes them generally more readable. Their added size also allows the inclusion of text labels indicating the command to which the icon corresponds. Once the meaning of each icon has become learned the icon can serve as a visual mnemonic, but until then the text label clearly and unambiguously relays the function the button will initiate.

In the right situation they can be a vital part of a quality user interface. The folder is a particularly useful and successful metaphor.

Intuitiveness

It is generally perceived that the most fundamental quality of any good user interface should be that it is intuitive. The problem is that "intuitive" means different things to different people. To some an intuitive user interface is one that users can figure out for themselves. There are some instances where this is helpful, but generally the didactic elements geared for the first-time user will hamper the effectiveness of intermediate or advanced users.

A much better definition of an intuitive user interface is one that is easy to learn. This does not mean that no instruction is required, but that it is minimal and that users can "pick it up" quickly and easily. First-time users might not intuit how to operate a scroll bar, but once it is explained they generally find it to be an intuitive idiom.

Icons, when clearly unambiguous, can help to make a user interface intuitive. But the user interface designer should never overlook the usefulness of good old-fashioned text labels.

Labels should be concise, cogent, and unambiguous. A good practice is to make labels conform to the terminology of the business that the application supports. This is a good way to pack a lot of meaning into a very few words.

Designing intuitive user interfaces is far more an art than a science. It draws more upon skills of psychology and cognitive reasoning than computer engineering or even graphic design. The process of Usability Testing, however, can assess the intuitiveness of a user interface in an objective manner.

Consistency

The standard GUI design elements go a long way to bring a level of consistency to every panel, but "look and feel" issues must be considered as well. The use of labels and icons must always be consistent. The same label or icon should always mean the same thing, and conversely the same thing should always be represented by the same label or icon.

User interface designers should always provide permanent objects as unchanging reference points around which the users can navigate. If they ever get lost or disoriented, they should be able to quickly find the permanent objects and from there get to where they need to be.

Simplicity

The most graceful solution to any problem is the one which is the most simple. The fewer things users have to see and do in order to get their work done, the happier and more effective they will be. A pitfall that should be avoided is "featuritis," providing an over-abundance of features that do not add value to the user interface. Features should not be included on a user interface unless there is a compelling need for them and they add significant value to the application.

Prevention

A fundamental tenet of graphic user interfaces is that it is preferable to prevent users from performing an inappropriate task in the first place rather than allowing the task to be performed and presenting a message afterwards saying that it couldn't be done. This is accomplished by disabling, or "graying out" certain elements under certain conditions.

Aesthetics

Finally, it is important that a user interface be aesthetically pleasing. It is possible for a user interface to be intuitive, easy to use, and efficient and still not be terribly nice to look at. While aesthetics do not directly impact the effectiveness of a user interface, users will be happier and therefore more productive if they are presented with an attractive user interface.

User Interface Design Elements

The Palette

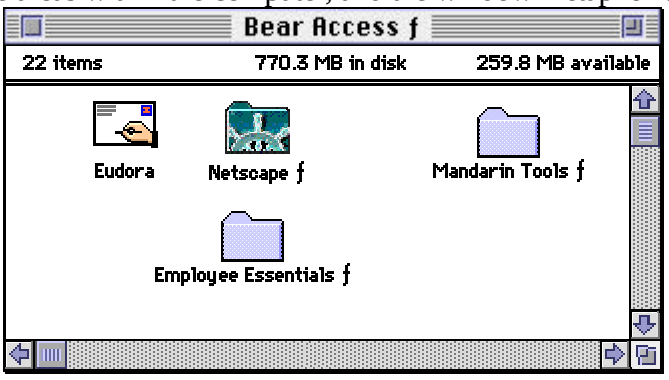
- Windows
- Events

The Paints

- Pull-down Menus / Drop-down Menus
- Push Buttons
- Icons
- Checkboxes
- Radio Buttons
- Scrolling Lists
- Text Field
- Popup List
- Spin Boxes
- Sliders

Windows

The most pervasive element used in GUIs is the window. It could be considered to be a metaphor for a "window" into the computer, but it is dependent on idioms for its operation. The video screen itself can be considered to be a window into the computer. In character-based interfaces it was the only window into the computer and was not really thought of as such. The GUI paradigm, however, allowed for the user to see into multiple areas within the computer, and the window metaphor was born.

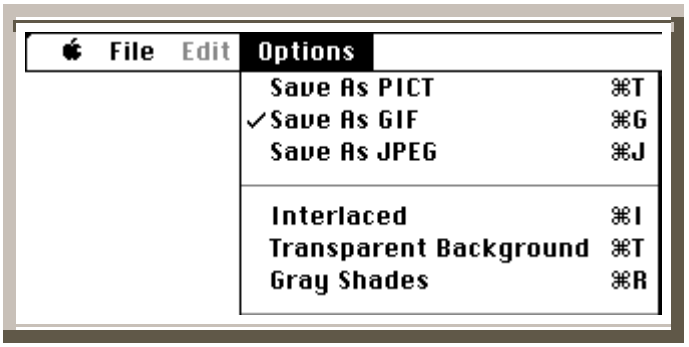


Events

An "event," with respect to user interfaces, is any function initiated by the user. Selecting something from a pull-down menu, clicking a button or a checkbox, and closing a window are all examples of events.

Pull-down Menus / Drop-down Menus

Pull-down menus are menus that the user can "pull down" from the menu bar that traverses the top of the screen. On some platforms these are called "drop down" menus because the user does not need to hold the mouse button down in order for the menu to remain visible.



Push Buttons

A push button is simply a rectangle that appears on a panel with some sort of label or icon inside it. The metaphor is to any button you'd find in the physical world, such as on a calculator or telephone. Clicking on a push button will cause some sort of action will occur. Sometimes a panel will have a "default" button, which appears with an enhanced border. This button will be activated when the Enter key is pressed.

Icons

Icons are small pictures that are generally represent objects in the physical world or are used as metaphors for functions or actions. Icons can be "clickable" and used to initiate an event. These are sometimes referred to as "buttcons," as they become hybrids of buttons and icons.

Checkboxes

A checkbox is a small square with some sort of label beside it. Clicking on a checkbox will cause an 'X' to appear in the box. Clicking it again will cause the 'X' to disappear.

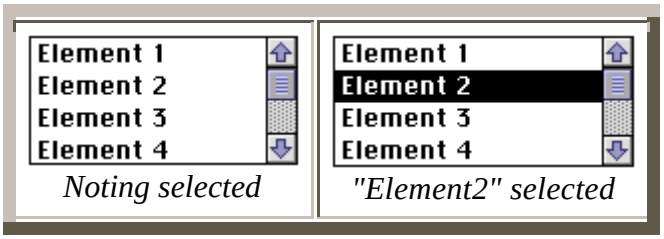
Radio Buttons

A radio button is small circle with some sort of label beside it. A black dot inside the circle indicates that the button is selected. The absence of a dot indicates that the button is unselected. With the radio button form object, clicking on one option will cause whatever other option that had been selected to become unselected.

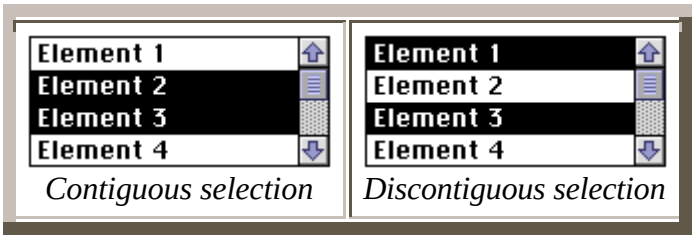
- When applicable, radio buttons should be placed in a logical order.
- Don't use radio buttons for binary choices (e.g. Yes/No). Use check boxes for this.

Scrolling Lists

Scrolling lists are lists of elements that appear in a box with a scroll bar on the side, allowing the user to scroll through the elements in the list. In this way the list can contain more elements than can be displayed at any one time. Clicking on an element in the list will cause that element to become selected, as shown below.



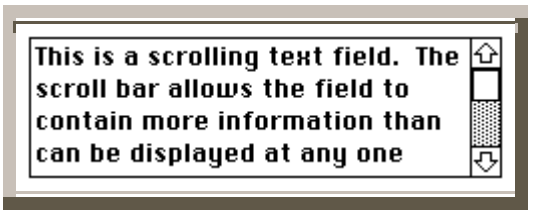
Optionally, a list can allow multiple selections. These can either be contiguous or discontiguous, as shown below (respectively).



Text Field

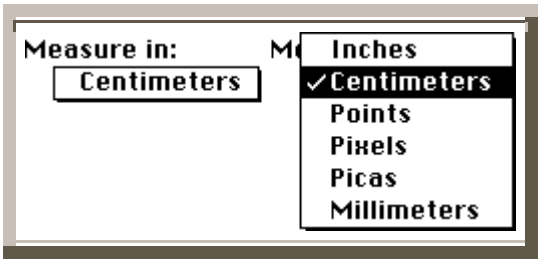
A text field is simply a space in which the user can type text. Text fields are usually contained within a rectangle, but it could just be space on a panel that can accept text.

If a text field is contained within a rectangle, it can optionally have a scroll bar to allow it to contain more information than can be displayed at any one time.



Popup List

A popup list appears initially as a box containing some sort of label. When the user clicks on the box, a larger box containing a variety of choices will "pop up." By moving the mouse, the user can cause another choice to become selected. When the user releases the mouse, the popup menu will disappear leaving currently selected choice to appear in the box.

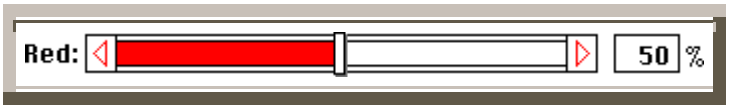


Spin Boxes

A spin box is a box containing some value, with up and down arrows on one side and generally a label on the other. By clicking the up or down arrow, different values will appear in the box. The entire range of choices can be cycled through if the up or down arrows are clicked enough times. Whatever value appears in the box is the value that is selected at that time.

Sliders

A slider is a long box with a control that the user is able to slide one way or the other. Often a slider will have an arrow button at either end to allow the user to adjust the slider one unit at a time. Also there is often a text field to the side of the slider that will indicate the value that the slider is currently set on and allow the user to enter a specific value.



Designing the User Interface

Display Things to the User

- Display a little text
- Display a lot of text
- Display a list of objects
- Display a hierarchical list of objects
- Display a warning, confirmation, or other brief message that presents a limited variety of actions

---

Where To Begin

It is always good to start with the primary goals of the user interface. This will reveal whether the focus should be on didactic qualities or efficiency. Once the goal is clearly established, a good approach is to focus on the specific tasks that the user interface will be required to perform in order to fulfill the goal. Then it's only a matter of deciding which design elements will best perform these tasks.

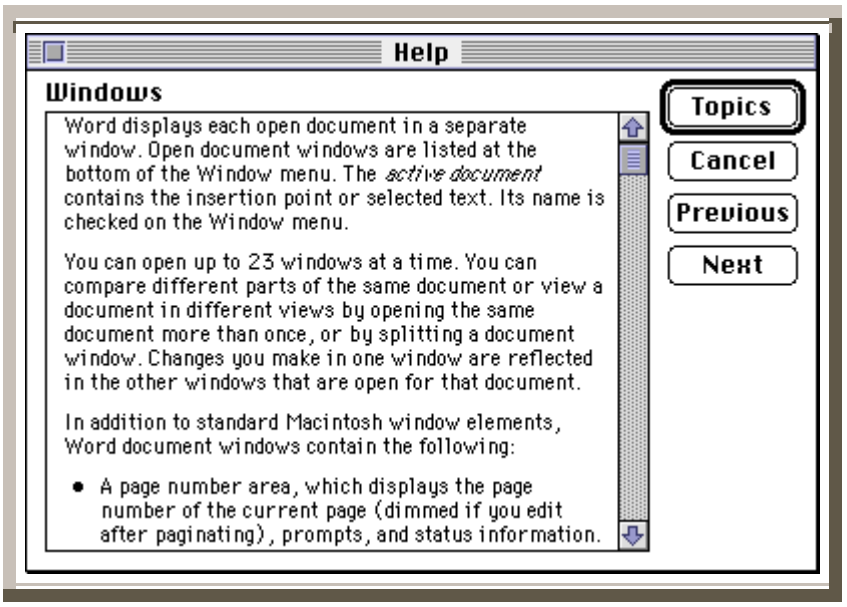
An enormous portion of the job of any user interface is to display information to the user. One could argue that this is the entire reason that computers exist in the first place.

Display a little text

This can easily be accomplished by simply displaying the text anywhere on the panel. Often it is accompanied by an icon of some sort to associate the text with a function or physical object.

Display a lot of text

If the space in the panel permits, the text can simply be displayed. But if the amount of text exceeds the space available, it can be placed in a scrolling text field. Often considerably large amounts of text will be broken down by topic or category, as in the example of the Microsoft Word help dialog shown below.

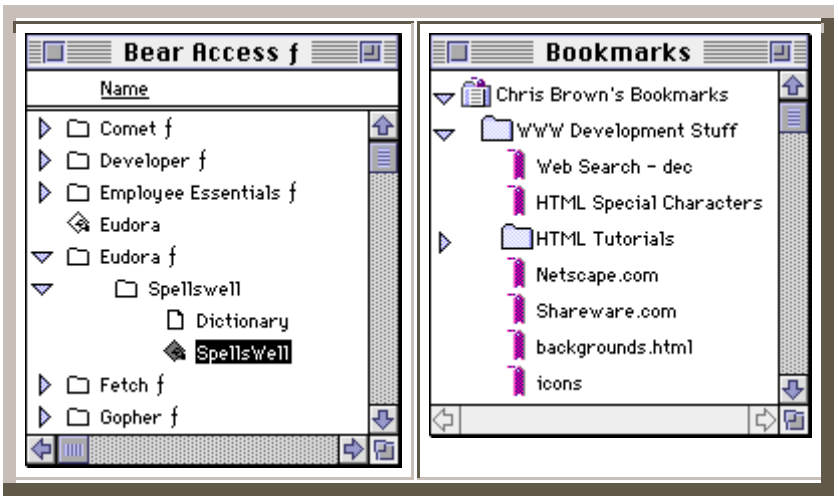


Display a list of objects

As with text, the presentation of a list of objects is generally determined by the size of the list and the amount of space available on the panel. If the space permits, the list can just be displayed as if it were text. But if the size of the list exceeds the space available, it can be displayed in a scrolling list.

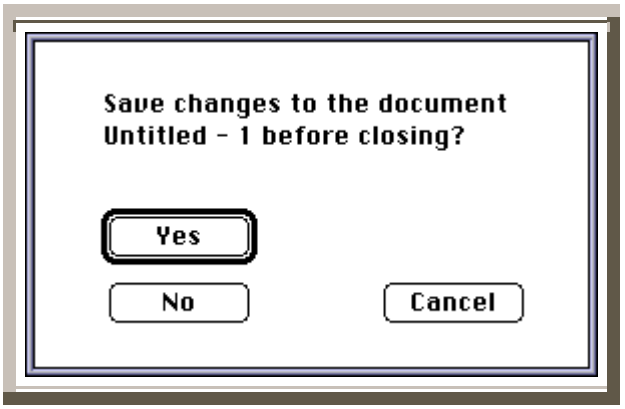
Display a hierarchical list of objects

This can be accomplished, as in the example from the Macintosh Finder below, by placing a combination of icons and text in a scrolling window. The same technique was used by Netscape for presenting bookmarks in their 2.0 version.



Display a warning, confirmation, or other brief message that presents a limited variety of actions

This can be accomplished by opening a dialog box and displaying the text of the message and a push button for each possible action.



Screen Design

Effective screen design guidelines

1. Screens should be attractive and uncluttered
2. Information on a single screen should be displayed in a meaningful, logical order
3. Screen designs should be consistent
4. Messages should be specific, understandable, and professional
5. Messages should remain on the screen for an appropriate period of time
6. Special effects should be used sparingly
7. Users should receive feedback
8. Screen designs should be documented and approved as soon as possible

Data entry screen design

Guidelines

1. Restrict user access to screen locations where data is entered
2. Provide a descriptive caption for each field and show the user where to enter the data
3. Show a sample format if one is required
4. Require ending keystroke for every field
5. Do not require users to enter special characters
6. Do not require users to type leading zeroes or trailing spaces for alphanumeric fields
7. Do not require users to type trailing zeroes that follow a decimal point
8. Display default values that users can accept
9. Use default values for constant data
10. Display a list of acceptable values for fields with a limited number of valid choices
11. Provide a way to leave the data entry screen without inputting the current record
12. Provide an opportunity to confirm the accuracy of input data before entering it
13. Provide a means to move among form fields in a standard, or in another, order
14. Design the screen form to match the layout of the source document
15. Allow the operator to add, change, delete, and view records
16. Design a method to allow operators to search for a specific record

Process control screen design

Users can control system actions with interactive menus and prompts

- Menu screens
- Menus display a list of user-selectable options
- Menu-driven system uses a hierarchy of main menus and submenus
- Shortcut key combinations can be used in a menu design
- Prompt screens
- User types a response to a prompt
- Responses can include commands
- Structured Query Language (SQL) can be used
- Question/answer screens can be used

- Natural language techniques can be used, similar to Internet search engines

Help screen design

Several methods to obtain Help

- Click a command button or toolbar
  - Press a special key
- Context-sensitive Help
  - Provides Help on the task in progress
- User-selected Help
- Hypertext
  - Uses links to display additional information on related topics
- Help Screen Design guidelines
  - Provide a direct route for users to return to the program after Help is obtained
- Title every Help screen
  - Use easy, simple, understandable Help text
- Present attractive, uncluttered screens
- Provide appropriate examples
- Use hyperlinks
- Include contact data for persons or departments responsible for assisting users

Source Document Design

Source documents

- Request and collect input data
- Can trigger or authorize input actions
- Provide a record of the original transaction

Form layout guidelines

- Allow sufficient space
- Offer clear instructions
- Provide logical organization
- Use captions effectively

Form zones

- Heading zone
- Control zone
- Instruction zone
- Totals zone
- Authorization zone

Source documents can be external or internal

Input Record Design

- Input record layout chart
- To design and document batch input records

- Multiple record designs are used for transactions that involve constant and repeating data
- Constant fields (non-repeating data)
- Repeating fields

Information flow on a form

- Should be logical and easy to follow
- Poor design results in forms that are difficult to use, time-consuming, and prone to error