Object Oriented Design

Objectives

- To explain how a software design may be represented as a set of interacting objects that manage their own state and operations
- To describe the activities in the object-oriented design process
- To introduce various models that describe an object-oriented design
- To show how the UML may be used to represent these models

Topics covered

- Objects and object classes
- An object-oriented design process
- Design evolution

Tutorial Questions

- Sommerville (6th Edition) Chapter 12 & 15
  - Question 12.1
  - Question 12.2
  - Question 12.4
  - Question 12.7
  - Question 15.1
  - Question 15.2
  - Question 15.10
Characteristics of OOD

- Objects are abstractions of real-world or system entities and manage themselves.
- Objects are independent and encapsulate state and representation information.
- System functionality is expressed in terms of object services.
- Shared data areas are eliminated. Objects communicate by message passing.
- Objects may be distributed and may execute sequentially or in parallel.

Advantages of OOD

- Easier maintenance. Objects may be understood as stand-alone entities.
- Objects are appropriate reusable components.
- For some systems, there may be an obvious mapping from real-world entities to system objects.

Object-oriented development

- Object-oriented analysis, design, and programming are related but distinct.
- OOA is concerned with developing an object model of the application domain.
- OOD is concerned with developing an object-oriented system model to implement requirements.
- OOP is concerned with realising an OOD using an OO programming language such as Java or C++.

Objects and object classes

- Objects are entities in a software system which represent instances of real-world and system entities.
- Object classes are templates for objects. They may be used to create objects.
- Object classes may inherit attributes and services from other object classes.

Objects

An object is an entity which has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.

Objects are created according to some object class definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.
The Unified Modeling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s
- The Unified Modeling Language is an integration of these notations
- It describes notations for a number of different models that may be produced during OO analysis and design
- It is now a de facto standard for OO modelling

Employee object class (UML)

Object communication

- Conceptually, objects communicate by message passing.
- Messages
  - The name of the service requested by the calling object.
  - Copies of the information required to execute the service and the name of a holder for the result of the service.
- In practice, messages are often implemented by procedure calls
  - Name = procedure name.
  - Information = parameter list.

Message examples

  // Call a method associated with a buffer
  // object that returns the next value
  // in the buffer
  v = circularBuffer.Get() ;

  // Call the method associated with a thermostat object that sets the
  // temperature to be maintained
  thermostat.setTemp(20) ;

Generalisation and inheritance

- Objects are members of classes which define attribute types and operations
- Classes may be arranged in a class hierarchy where one class (a super-class) is a generalisation of one or more other classes (sub-classes)
- A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own
- Generalisation in the UML is implemented as inheritance in OO programming languages

A generalisation hierarchy
Advantages of inheritance

- It is an abstraction mechanism which may be used to classify entities
- It is a reuse mechanism at both the design and the programming level
- The inheritance graph is a source of organisational knowledge about domains and systems

Problems with inheritance

- Object classes are not self-contained; they cannot be understood without reference to their super-classes
- Designers have a tendency to reuse the inheritance graph created during analysis. This can lead to significant inefficiency
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained

Inheritance and OOD

- There are differing views as to whether inheritance is fundamental to OOD.
  - View 1. Identifying the inheritance hierarchy or network is a fundamental part of object-oriented design. Obviously this can only be implemented using an OOPL.
  - View 2. Inheritance is a useful implementation concept which allows reuse of attribute and operation definitions. Identifying an inheritance hierarchy at the design stage places unnecessary restrictions on the implementation
- Inheritance introduces complexity and this is undesirable, especially in critical systems

UML associations

- Objects and object classes participate in relationships with other objects and object classes
- In the UML, a generalised relationship is indicated by an association
- Associations may be annotated with information that describes the association
- Associations are general but may indicate that an attribute of an object is an associated object or that a method relies on an associated object

An association model

Employee is-member-of Department
is-managed-by
manages Manager

Concurrent objects

- The nature of objects as self-contained entities make them suitable for concurrent implementation
- The message-passing model of object communication can be implemented directly if objects are running on separate processors in a distributed system
Servers and active objects

- Servers.
  - The object is implemented as a parallel process (server) with entry points corresponding to object operations. If no calls are made to it, the object suspends itself and waits for further requests for service.
- Active objects
  - Objects are implemented as parallel processes and the internal object state may be changed by the object itself and not simply by external calls.

Active transponder object

- Active objects may have their attributes modified by operations but may also update them autonomously using internal operations.
- Transponder object broadcasts an aircraft’s position. The position may be updated using a satellite positioning system. The object periodically update the position by triangulation from satellites.

An active transponder object

```java
class Transponder extends Thread {
    Position currentPosition;
    Coords c1, c2;
    Satellite sat1, sat2;
    Navigator theNavigator;

    public Position givePosition () {
        return currentPosition;
    }

    public void run () {
        while (true) {
            c1 = sat1.position ();
            c2 = sat2.position ();
            currentPosition = theNavigator.compute (c1, c2);
        }
    }
}
```

Java threads

- Threads in Java are a simple construct for implementing concurrent objects.
- Threads must include a method called run() and this is started up by the Java run-time system.
- Active objects typically include an infinite loop so that they are always carrying out the computation.

An object-oriented design process

- Define the context and modes of use of the system.
- Design the system architecture.
- Identify the principal system objects.
- Develop design models.
- Specify object interfaces.

Weather system description

A weather data collection system is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellites. Weather stations transmit their data to the area computer in response to a request from that machine.

The area computer validates the collected data and integrates it with the data from different sources. The integrated data is archived and, using data from this archive and a digitised map database, a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer or may be displayed in a number of different formats.
Weather station description

A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected every five minutes.

When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

Layered architecture

- Data display layer where objects are concerned with presenting the data in a human-readable form.
- Data processing layer where objects are concerned with storing the data for future processing.
- Data archiving layer where objects are concerned with acquiring data from remote sources.
- Data display layer where objects are concerned with checking and integrating the collected data.

System context and models of use

- Develop an understanding of the relationships between the software being designed and its external environment.
- System context
  - A static model that describes other systems in the environment. Use a subsystem model to show other systems. Following slide shows the systems around the weather station system.
- Model of system use
  - A dynamic model that describes how the system interacts with its environment. Use use-cases to show interactions.

Subsystems in the weather mapping system

- Weather station
- Satellite
- Comms
- Balloon
- Observer

Use-case description

- **System**: Weather station
- **Use-case**: Report
- **Actors**: Weather data collection system, Weather station
- **Data**: The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather data collection system. The data sent are the maximum minimum and average ground and air temperatures, the maximum, minimum and average air pressures, the maximum, minimum and average wind speeds, the total rainfall and the wind direction as sampled at 5 minute intervals.
- **Stimulus**: The weather data collection system establishes a modem link with the weather station and requests transmission of the data.
- **Response**: The summarised data is sent to the weather data collection system.
- **Comments**: Weather stations are usually asked to report once per hour but this frequency may differ from one station to the other and may be modified in future.
Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- Layered architecture is appropriate for the weather station:
  - Interface layer for handling communications
  - Data collection layer for managing instruments
  - Instruments layer for collecting data
- There should be no more than 7 entities in an architectural model.

Weather station architecture

Object identification

- Identifying objects (or object classes) is the most difficult part of object-oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience, and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

Approaches to identification

- Use a grammatical approach based on a natural language description of the system (used in Hood method).
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes, and methods in each scenario are identified.

Weather station object classes

- Ground thermometer, Anemometer, Barometer
  - Application domain objects that are 'hardware' objects related to the instruments in the system.
- Weather station
  - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
- Weather data
  - Encapsulates the summarised data from the instruments.

Weather station object classes

- WeatherStation
  - identifier
  - reportWeather(): calibrate(instruments), test(): startup(instruments), shutdown(instruments)
- WeatherData
  - airTemperatures, groundTemperatures, windSpeeds, windDirections, pressures, rainfall
  - collect(): summarise()
- GroundThermometer
  - temperature(): calibrate()
- Anemometer
  - windSpeed(): windDirection(): test()
- Barometer
  - pressure(): height()
Further objects and object refinement

- Use domain knowledge to identify more objects and operations
  - Weather stations should have a unique identifier
  - Weather stations are remotely situated so instrument failures have to be reported automatically. Therefore attributes and operations for self-checking are required
- Active or passive objects
  - In this case, objects are passive and collect data on request rather than autonomously. This introduces flexibility at the expense of controller processing time

Design models

- Design models show the objects and object classes and relationships between these entities
- Static models describe the static structure of the system in terms of object classes and relationships
- Dynamic models describe the dynamic interactions between objects.

Examples of design models

- Sub-system models that show logical groupings of objects into coherent subsystems
- Sequence models that show the sequence of object interactions
- State machine models that show how individual objects change their state in response to events
- Other models include use-case models, aggregation models, generalisation models, etc.

Subsystem models

- Shows how the design is organised into logically related groups of objects
- In the UML, these are shown using packages - an encapsulation construct. This is a logical model. The actual organisation of objects in the system may be different.

Weather station subsystems

- Sequence models show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top
  - Time is represented vertically so models are read top to bottom
  - Interactions are represented by labelled arrows. Different styles of arrow represent different types of interaction
  - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system
### Data collection sequence

- CommsController
- WeatherStation
- WeatherData

**Request**
- CommsController
  - request (report)
  - acknowledge ()
  - reply (report)
  - acknowledge ()
  - send (report)

**WeatherStation**
- report ()
- summarise ()
- calibrate ()
- test ()
- shutdown ()

**WeatherData**
- calibrate ()
- test ()
- startup ()
- shutdown ()
- reportWeather ()

### Statecharts

- Show how objects respond to different service requests and the state transitions triggered by these requests
  - If object state is Shutdown then it responds to a Startup() message
  - In the waiting state the object is waiting for further messages
  - If reportWeather () then system moves to summarising state
  - If calibrate () the system moves to a calibrating state
  - A collecting state is entered when a clock signal is received

### Weather station state diagram

- Shutdown
- Waiting
- Testing
- Transmitting
- Collecting
- Summarising
- Calibrating

**Operation**
- reportWeather ( )
- test ()
- startup ()
- shutdown ()

**Calibration**
- calibration OK
- test complete

**Transmission**
- collection complete
- weather summary complete

**Interfaces**

- interface WeatherStation { public void WeatherStation () ; public void startup () ; public void shutdown () ; public void reportWeather () ; public void test () ; public void calibrate () ; public int getID () ; }

### Object interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel
- Designers should avoid designing the interface representation but should hide this in the object itself
- Objects may have several interfaces which are viewpoints on the methods provided
- The UML uses class diagrams for interface specification but Java may also be used

### Design evolution

- Hiding information inside objects means that changes made to an object do not affect other objects in an unpredictable way
- Assume pollution monitoring facilities are to be added to weather stations. These sample the air and compute the amount of different pollutants in the atmosphere
- Pollution readings are transmitted with weather data
Changes required

- Add an object class called ‘Air quality’ as part of WeatherStation
- Add an operation reportAirQuality to WeatherStation. Modify the control software to collect pollution readings
- Add objects representing pollution monitoring instruments

Key points

- OOD is an approach to design so that design components have their own private state and operations
- Objects should have constructor and inspection operations. They provide services to other objects
- Objects may be implemented sequentially or concurrently
- The Unified Modeling Language provides different notations for defining different object models

User Interface Design


User interface design

- Designing effective interfaces for software systems
Objectives

- To suggest some general design principles for user interface design
- To explain different interaction styles
- To introduce styles of information presentation
- To describe the user support which should be built-in to user interfaces
- To introduce usability attributes and system approaches to system evaluation

Topics covered

- User interface design principles
- User interaction
- Information presentation
- User support
- Interface evaluation

The user interface

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

Graphical user interfaces

- Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used

GUI characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Multiple windows allow different information to be displayed simultaneously on the user’s screen.</td>
</tr>
<tr>
<td>Icons</td>
<td>Icons represent different types of information. On some systems, icons represent files; on others, icons represent processes.</td>
</tr>
<tr>
<td>Menus</td>
<td>Commands are selected from a menu rather than typed in a command language.</td>
</tr>
<tr>
<td>Pointing</td>
<td>A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window.</td>
</tr>
<tr>
<td>Graphics</td>
<td>Graphical elements can be mixed with text on the same display.</td>
</tr>
</tbody>
</table>

GUI advantages

- They are easy to learn and use.
  - Users without experience can learn to use the system quickly.
- The user may switch quickly from one task to another and can interact with several different applications.
  - Information remains visible in its own window when attention is switched.
- Fast, full-screen interaction is possible with immediate access to anywhere on the screen.
User-centred design

• The aim of this chapter is to sensitise software engineers to key issues underlying the design rather than the implementation of user interfaces
• User-centred design is an approach to UI design where the needs of the user are paramount and where the user is involved in the design process
• UI design always involves the development of prototype interfaces

UI design principles

• UI design must take account of the needs, experience and capabilities of the system users
• Designers should be aware of people’s physical and mental limitations (e.g. limited short-term memory) and should recognise that people make mistakes
• UI design principles underlie interface designs although not all principles are applicable to all designs

Design principles

• User familiarity
  – The interface should be based on user-oriented terms and concepts rather than computer concepts. For example, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
• Consistency
  – The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.
• Minimal surprise
  – If a command operates in a known way, the user should be able to predict the operation of comparable commands

User interface design process

• User interface design principles
  – User familiarity: The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
  – Consistency: The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
  – Minimal surprise: Users should never be surprised by the behaviour of a system.
  – Recoverability: The interface should include mechanisms to allow users to recover from errors.
  – User guidance: The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
  – User diversity: The interface should provide appropriate interaction facilities for different types of system user.
User-system interaction

- Two problems must be addressed in interactive systems design
  - How should information from the user be provided to the computer system?
  - How should information from the computer system be presented to the user?
- User interaction and information presentation may be integrated through a coherent framework such as a user interface metaphor.

Interaction styles

- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

Advantages and disadvantages

<table>
<thead>
<tr>
<th>Interaction style</th>
<th>Main advantages</th>
<th>Main disadvantages</th>
<th>Application examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct manipulation</td>
<td>Fast and intuitive interaction</td>
<td>May be hard to implement</td>
<td>Video games, CAD systems</td>
</tr>
<tr>
<td>Menu selection</td>
<td>Accurate user input</td>
<td>May be hard to learn</td>
<td>Most general-purpose systems</td>
</tr>
<tr>
<td>Form fill-in</td>
<td>Simple and quick</td>
<td>May be complex</td>
<td>Stock control, personal loan processing</td>
</tr>
<tr>
<td>Command language</td>
<td>Powerful and flexible</td>
<td>Hard to learn</td>
<td>Operating systems, library information retrieval systems</td>
</tr>
<tr>
<td>Natural language</td>
<td>Accessible to casual users</td>
<td>Requires more typing</td>
<td>Timetable systems, WWW information retrieval systems</td>
</tr>
</tbody>
</table>

Direct manipulation problems

- The derivation of an appropriate information space model can be very difficult
- Given that users have a large information space, what facilities for navigating around that space should be provided?
- Direct manipulation interfaces can be complex to program and make heavy demands on the computer system.
**Menu systems**

- Users make a selection from a list of possibilities presented to them by the system.
- The selection may be made by pointing and clicking with a mouse, using cursor keys or by typing the name of the selection.
- May make use of simple-to-use terminals such as touchscreens.

**Advantages of menu systems**

- Users need not remember command names as they are always presented with a list of valid commands.
- Typing effort is minimal.
- User errors are trapped by the interface.
- Context-dependent help can be provided. The user’s context is indicated by the current menu selection.

**Problems with menu systems**

- Actions which involve logical conjunction (and) or disjunction (or) are awkward to represent.
- Menu systems are best suited to presenting a small number of choices. If there are many choices, some menu structuring facility must be used.
- Experienced users find menus slower than command language.

**Form-based interface**

<table>
<thead>
<tr>
<th>Title</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Price</td>
</tr>
<tr>
<td>Publisher</td>
<td>Publication Date</td>
</tr>
<tr>
<td>Edition</td>
<td>Number of copies</td>
</tr>
<tr>
<td>Classification</td>
<td>Loan status</td>
</tr>
<tr>
<td>Date of purchase</td>
<td>Order status</td>
</tr>
</tbody>
</table>

**Command interfaces**

- User types commands to give instructions to the system e.g. UNIX.
- May be implemented using cheap terminals.
- Easy to process using compiler techniques.
- Commands of arbitrary complexity can be created by command combination.
- Concise interfaces requiring minimal typing can be created.

**Problems with command interfaces**

- Users have to learn and remember a command language. Command interfaces are therefore unsuitable for occasional users.
- Users make errors in command. An error detection and recovery system is required.
- System interaction is through a keyboard so typing ability is required.
Command languages

- Often preferred by experienced users because they allow for faster interaction with the system
- Not suitable for casual or inexperienced users
- May be provided as an alternative to menu commands (keyboard shortcuts). In some cases, a command language interface and a menu-based interface are supported at the same time.

Natural language interfaces

- The user types a command in a natural language. Generally, the vocabulary is limited and these systems are confined to specific application domains (e.g. timetable enquiries)
- NL processing technology is now good enough to make these interfaces effective for casual users but experienced users find that they require too much typing.

Multiple user interfaces

- Information presentation is concerned with presenting system information to system users
- The information may be presented directly (e.g. text in a word processor) or may be transformed in some way for presentation (e.g. in some graphical form)
- The Model-View-Controller approach is a way of supporting multiple presentations of data.
**Information presentation**

- **Static information**
  - Initialised at the beginning of a session. It does not change during the session.
  - May be either numeric or textual.

- **Dynamic information**
  - Changes during a session and the changes must be communicated to the system user.
  - May be either numeric or textual.

**Information display factors**

- Is the user interested in precise information or data relationships?
- How quickly do information values change? Must the change be indicated immediately?
- Must the user take some action in response to a change?
- Is there a direct manipulation interface?
- Is the information textual or numeric? Are relative values important?

**Alternative information presentations**

- [Graph showing data for January to June with bar charts and line graphs.]

**Analogue vs. digital presentation**

- **Digital presentation**
  - Compact - takes up little screen space.
  - Precise values can be communicated.

- **Analogue presentation**
  - Easier to get an ‘at a glance’ impression of a value.
  - Possible to show relative values.
  - Easier to see exceptional data values.

**Dynamic information display**

- Dial with needle
- Pie chart
- Thermometer
- Horizontal bar

**Displaying relative values**

- Pressure
  - [Graph showing pressure values with a horizontal bar chart and a thermometer.]

- Temperature
  - [Graph showing temperature values with a horizontal bar chart and a thermometer.]
Textual highlighting

The filename you have chosen has been used. Please choose another name.

Ch. 16 User interface design

OK Cancel

Data visualisation

- Concerned with techniques for displaying large amounts of information
- Visualisation can reveal relationships between entities and trends in the data
- Possible data visualisations are:
  - Weather information collected from a number of sources
  - The state of a telephone network as a linked set of nodes
  - Chemical plant visualised by showing pressures and temperatures in a linked set of tanks and pipes
  - A model of a molecule displayed in 3 dimensions
  - Web pages displayed as a hyperbolic tree

Colour displays

- Colour adds an extra dimension to an interface and can help the user understand complex information structures
- Can be used to highlight exceptional events
- Common mistakes in the use of colour in interface design include:
  - The use of colour to communicate meaning
  - Over-use of colour in the display

Colour use guidelines

- Don't use too many colours
- Use colour coding to support use tasks
- Allow users to control colour coding
- Design for monochrome then add colour
- Use colour coding consistently
- Avoid colour pairings which clash
- Use colour change to show status change
- Be aware that colour displays are usually lower resolution

User support

- User guidance covers all system facilities to support users including on-line help, error messages, manuals etc.
- The user guidance system should be integrated with the user interface to help users when they need information about the system or when they make some kind of error
- The help and message system should, if possible, be integrated
**Error messages**

- Error message design is critically important. Poor error messages can mean that a user rejects rather than accepts a system.
- Messages should be polite, concise, consistent and constructive.
- The background and experience of users should be the determining factor in message design.

**Design factors in message wording**

<table>
<thead>
<tr>
<th>Context</th>
<th>The user guidance system should be aware of what the user is doing and should adjust the output message to the context.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience</td>
<td>As users become familiar with a system, they become impatient with long, ‘meaningful’ messages. However, beginners find it difficult to understand short terse statements of the problem. The user guidance system should provide both types of message and allow the user to control message contents.</td>
</tr>
<tr>
<td>Skill level</td>
<td>Messages should be tailored to the user’s skills as well as their experience. Messages for different classes of user may be expressed in different ways depending on the terminology which is familiar to the user.</td>
</tr>
<tr>
<td>Style</td>
<td>Messages should be positive rather than negative. They should use the active rather than the passive mode of address. They should never be insulting or try to be funny.</td>
</tr>
<tr>
<td>Culture</td>
<td>Wherever possible, the language of messages should be familiar with the culture of the country where the system is sold. There are distinct cultural differences between Europe, Asia and America. A suitable message for one culture might be unacceptable in another.</td>
</tr>
</tbody>
</table>

**Nurse input of a patient’s name**

Please type the patient name in the box then click OK

Bates J

**System and user-oriented error messages**

System-oriented error message

- Error #27
- Invalid patient id entered

User-oriented error message

- Patient J. Bates is not registered
- Click on Patients for a list of registered patients
- Click on Retry to re-input a patient name
- Click on Help for more information

**Help system design**

- *Help?* means ‘help I want information”
- *Help!* means “HELP. I’m in trouble”
- Both of these requirements have to be taken into account in help system design
- Different facilities in the help system may be required

**Help information**

- Should not simply be an on-line manual
- Screens or windows don’t map well onto paper pages.
- The dynamic characteristics of the display can improve information presentation.
- People are not so good at reading screen as they are text.
Help system use

- Multiple entry points should be provided so that the user can get into the help system from different places.
- Some indication of where the user is positioned in the help system is valuable.
- Facilities should be provided to allow the user to navigate and traverse the help system.

Entry points to a help system

- Multiple entry points should be provided so that the user can get into the help system from different places.
- Some indication of where the user is positioned in the help system is valuable.
- Facilities should be provided to allow the user to navigate and traverse the help system.

Help system windows

- Mail redirection
  - Mail may be redirected to another network address by pressing the redirect button in the control panel. The system is set for the name of the user or users to whom the mail has been sent.

User documentation

- As well as on-line information, paper documentation should be supplied with a system.
- Documentation should be designed for a range of users from inexperienced to experienced.
- As well as manuals, other easy-to-use documentation such as a quick reference card may be provided.

User document types

- Functional description
  - Brief description of what the system can do
- Introductory manual
  - Presents an informal introduction to the system
- System reference manual
  - Describes all system facilities in detail
- System installation manual
  - Describes how to install the system
- System administrator’s manual
  - Describes how to manage the system when it is in use
User interface evaluation

- Some evaluation of a user interface design should be carried out to assess its suitability.
- Full scale evaluation is very expensive and impractical for most systems.
- Ideally, an interface should be evaluated against a usability specification. However, it is rare for such specifications to be produced.

Usability attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learnability</td>
<td>How long does it take a new user to become productive with the system?</td>
</tr>
<tr>
<td>Speed of operation</td>
<td>How well does the system response match the user’s work practice?</td>
</tr>
<tr>
<td>Robustness</td>
<td>How tolerant is the system of user error?</td>
</tr>
<tr>
<td>Recoverability</td>
<td>How good is the system at recovering from user errors?</td>
</tr>
<tr>
<td>Adaptability</td>
<td>How closely is the system tied to a single model of work?</td>
</tr>
</tbody>
</table>

Simple evaluation techniques

- Questionnaires for user feedback.
- Video recording of system use and subsequent tape evaluation.
- Instrumentation of code to collect information about facility use and user errors.
- The provision of a grip button for on-line user feedback.

Key points

- Interface design should be user-centred. An interface should be logical and consistent and help users recover from errors.
- Interaction styles include direct manipulation, menu systems form fill-in, command languages and natural language.
- Graphical displays should be used to present trends and approximate values. Digital displays when precision is required.
- Colour should be used sparingly and consistently.

Key points

- Systems should provide on-line help. This should include “help, I’m in trouble” and “help, I want information”.
- Error messages should be positive rather than negative.
- A range of different types of user documents should be provided.
- Ideally, a user interface should be evaluated against a usability specification.