DESIGN

Purpose of Design

- Design is where customer requirements, business needs, and technical considerations all come together in the formulation of a product or system

- The design model provides detail about the software data structures, architecture, interfaces, and components

- The design model can be assessed for quality and be improved before code is generated and tests are conducted

(More on next slide)

Purpose of Design

- Does the design contain errors, inconsistencies, or omissions?
- Are there better design alternatives?
- Can the design be implemented within the constraints, schedule, and cost that has been established?

Purpose of Design (continued)

- A designer must practice diversification and convergence
  - The designer selects from design components, component solutions, and knowledge available through catalogs, textbooks, and experience
  - The designer then chooses the elements from this collection that meet the requirements defined by requirements engineering and analysis modeling
  - Convergence occurs as alternatives are considered and rejected until one particular configuration of components is chosen

Purpose of Design (continued)

- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software
  - Design begins at a high level of abstraction that can be directly traced back to the data, functional, and behavioral requirements
  - As design iteration occurs, subsequent refinement leads to design representations at much lower levels of abstraction

From Analysis Model to Design Model

- Each element of the analysis model provides information that is necessary to create the four design models
  - The data/class design transforms analysis classes into design classes along with the data structures required to implement the software

  - The architectural design defines the relationship between major structural elements of the software; architectural styles and design patterns help achieve the requirements defined for the system

(More on next slide)

From Analysis Model to Design Model

- The interface design describes how the software communicates with systems that interoperate with it and with humans that use it

- The component-level design transforms structural elements of the software architecture into a procedural description of software components

From Analysis Model to Design Model (continued)

Component-level Design

(Class-based model, Flow-oriented model
Behavioral model)

Interface Design

(Scenario-based model, Flow-oriented model
Behavioral model)

Architectural Design

(Class-based model, Flow-oriented model)

Data/Class Design

(Class-based model, Behavioral model)

Task Set for Software Design

1) Examine the information domain model and design appropriate data structures for data objects and their attributes
2) Using the analysis model, select an architectural style (and design patterns) that are appropriate for the software
3) Partition the analysis model into design subsystems and allocate these subsystems within the architecture
    1) Design the subsystem interfaces
    2) Allocate analysis classes or functions to each subsystem

(More on next slide)

Task Set for Software Design

4) Create a set of design classes or components
   a) Translate each analysis class description into a design class

   b) Check each design class against design criteria; consider inheritance issues

   c) Define methods associated with each design class

   a) Evaluate and select design patterns for a design class or subsystem

Task Set for Software Design (continued)

5) Design any interface required with external systems or devices
6) Design the user interface
7) Conduct component-level design
   a) Specify all algorithms at a relatively low level of abstraction
   b) Refine the interface of each component

Task Set for Software Design (continued)

a) Define component-level data structures

b) Review each component and correct all errors uncovered

5) Develop a deployment model

a) Show a physical layout of the system, revealing which components will be located where in the physical computing environment

Design Quality

- The importance of design is quality

- Design is the place where quality is fostered
  - Provides representations of software that can be assessed for quality
  - Accurately translates a customer's requirements into a finished software product or system
  - Serves as the foundation for all software engineering activities that follow

Design Quality

- Without design, we risk building an unstable system that
  - Will fail when small changes are made
  - May be difficult to test
  - Cannot be assessed for quality later in the software process when time is short and most of the budget has been spent

- The quality of the design is assessed through a series of formal technical reviews or design walkthroughs

Goals of a Good Design

- The design must implement all of the explicit requirements contained in the analysis model
  - It must also accommodate all of the implicit requirements desired by the customer
- The design must be a readable and understandable guide for those who generate code, and for those who test and support the software
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implement perspective

"Writing a clever piece of code that works is one thing; designing something that can support a long-lasting business is quite another."

Design Quality Guidelines

1) A design should exhibit an <u>architecture</u> that
    a) Has been created using recognizable <u>architectural styles or patterns</u>

    b) Is composed of components that exhibit good design characteristics

    a) Can be implemented in an <u>evolutionary</u> fashion, thereby facilitating implementation and testing

(more on next slide)

Design Quality Guidelines

1) A design should be modular; that is, the software should be logically partitioned into elements or subsystems

1) A design should contain distinct representations of data, architecture, interfaces, and components

2) A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns

Quality Guidelines (continued)

5) A design should lead to <u>components</u> that exhibit <u>independent</u> functional characteristics
6) A design should lead to interfaces that <u>reduce the complexity of connections</u> between components and with the external environment
7) A design should be derived using a repeatable method that is <u>driven by</u> information obtained during software <u>requirements analysis</u>
8) A design should be represented using a <u>notation</u> that effectively communicates its meaning

> "Quality isn't something you lay on top of subjects and objects like tinsel on a Christmas tree."

Design Concepts

- Abstraction
  - Procedural abstraction – a sequence of instructions that have a specific and limited function
  - Data abstraction – a named collection of data that describes a data object
- Architecture
  - The overall structure of the software and the ways in which the structure provides conceptual integrity for a system
  - Consists of components, connectors, and the relationship between them

(more on next slide)

Design Concepts

- Patterns
  - A design structure that solves a particular design problem within a specific context
  - It provides a description that enables a designer to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns
- Modularity
  - Separately named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle)
  - Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity

Design Concepts (continued)

- Information hiding
  - The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules
  - This enforces access constraints to both procedural (i.e., implementation) detail and local data structures

- Functional independence
  - Modules that have a "single-minded" function and an aversion to excessive interaction with other modules
  - High cohesion – a module performs only a single task
  - Low coupling – a module has the lowest amount of connection needed with other modules

(more on next slide)

Design Concepts (continued)

- Stepwise refinement
  - Development of a program by <u>successively refining</u> levels of procedure detail
  - Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details
- Refactoring
  - A reorganization technique that <u>simplifies the design</u> (or internal code structure) of a component <u>without changing</u> its function or external behavior
  - Removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures

Design Concepts (continued)

- Design classes
  - Refines the analysis classes by providing design detail that will enable the classes to be implemented
  - Creates a new set of design classes that implement a software infrastructure to support the business solution

Types of Design Classes

- **User interface classes** – define all abstractions necessary for human-computer interaction (usually via metaphors of real-world objects)
- **Business domain classes** – refined from analysis classes; identify attributes and services (methods) that are required to implement some element of the business domain
- **Process classes** – implement lower-level business abstractions required to fully manage the business domain classes

Types of Design Classes

- **Persistent classes** – represent data stores (e.g., a database) that will persist beyond the execution of the software

- **System classes** – implement software management and control functions that enable the system to operate and communicate within its computing environment and the <u>outside world</u>

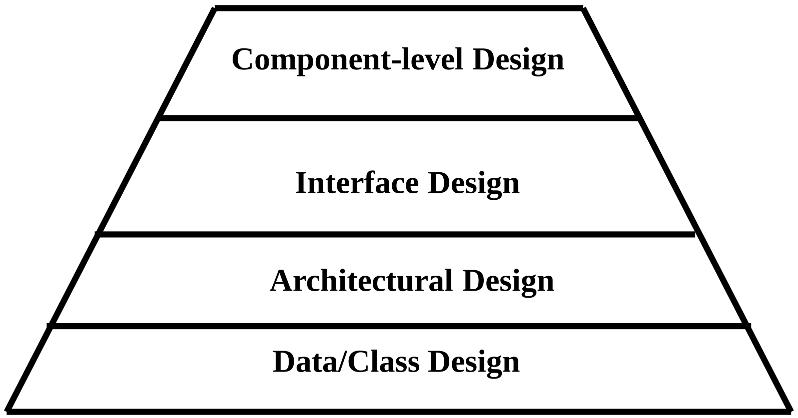
Characteristics of a Well-Formed Design Class

- Complete and sufficient
  - Contains the complete encapsulation of all attributes and methods that exist for the class
  - Contains only those methods that are sufficient to achieve the intent of the class
- Primitiveness
  - Each method of a class focuses on accomplishing one service for the class
  - Each class should have limited knowledge of other classes in other subsystems

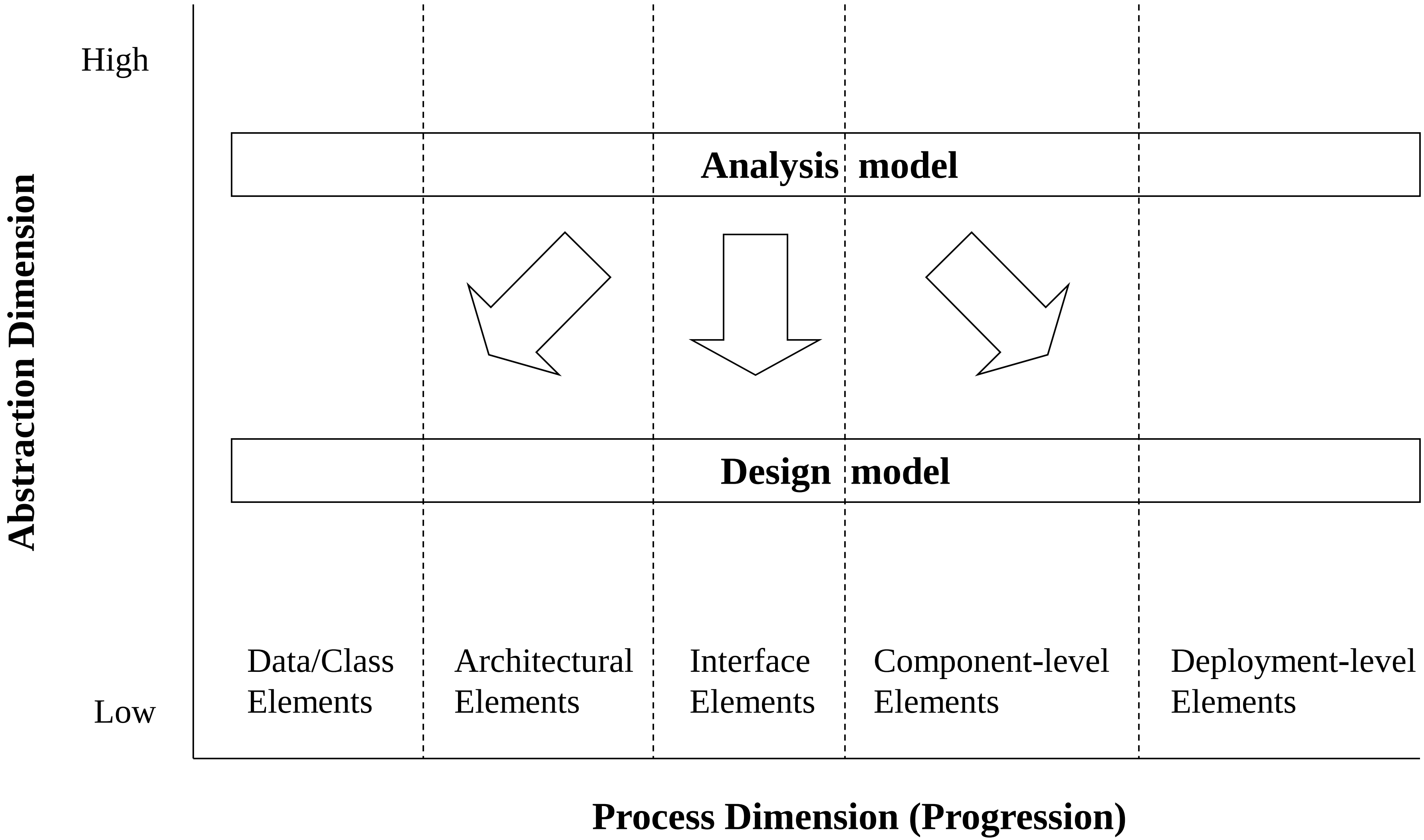
Characteristics of a Well-Formed Design Class

- High cohesion
  - The class has a small, focused set of responsibilities and single-mindedly applies attributes and methods to implement those responsibilities
- Low coupling
  - Collaboration of the class with other classes is kept to an acceptable minimum

The Design Model



Dimensions of the Design Model



Design Model

- The design model can be viewed in two different dimensions
  – (Horizontally) The process dimension indicates the evolution of the parts of the design model as each design task is executed

  – (Vertically) The abstraction dimension represents the level of detail as each element of the analysis model is transformed into the design model and then iteratively refined

- Elements of the design model use many of the same UML diagrams used in the analysis model
  – The diagrams are refined and elaborated as part of the design

(More on next slide)

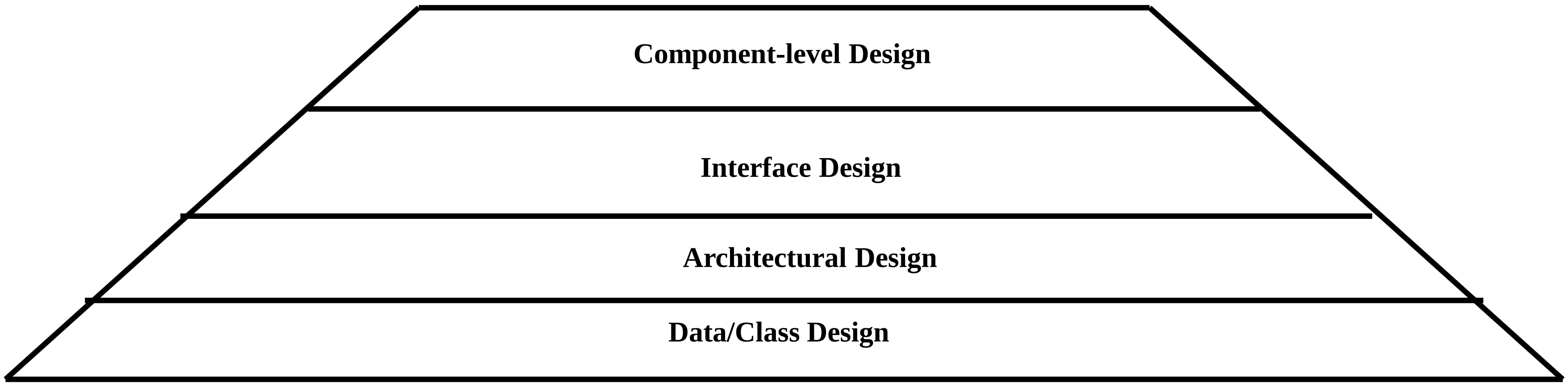
Design Model

  - More implementation-specific detail is provided
  - Emphasis is placed on
    - Architectural structure and style
    - Interfaces between components and the outside world
    - Components that reside within the architecture

- Design model elements are not always developed in a sequential fashion
  - Preliminary architectural design sets the stage
  - It is followed by interface design and component-level design, which often occur in parallel

Introduction (continued)

- The design model has the following layered elements
  - Data/class design
  - Architectural design
  - Interface design
  - Component-level design
- A fifth element that follows all of the others is deployment-level design



Design Elements

- Data/class design
  - Creates a model of data that is represented at a high level of abstraction
- Architectural design
  - Depicts the overall layout of the software
- Interface design
  - Tells how information flows into and out of the system and how it is communicated among the components defined as part of the architecture
  - Includes the user interface, external interfaces, and internal interfaces

Design Elements

- Component-level design elements
  - Describes the internal detail of each software component by way of data structure definitions, algorithms, and interface specifications

- Deployment-level design elements
  - Indicates how software functionality and subsystems will be allocated within the physical computing environment that will support the software

Pattern-based Software Design

- Mature engineering disciplines make use of thousands of design patterns for such things as buildings, highways, electrical circuits, factories, weapon systems, vehicles, and computers

- Design patterns also serve a purpose in software engineering
- Architectural patterns
  - Define the overall structure of software
  - Indicate the relationships among subsystems and software components
  - Define the rules for specifying relationships among software elements

Pattern-based Software Design

- Design patterns
  - Address a specific element of the design such as an aggregation of components or solve some design problem, relationships among components, or the mechanisms for effecting inter-component communication
  - Consist of creational, structural, and behavioral patterns
- Coding patterns
  - Describe language-specific patterns that implement an algorithmic or data structure element of a component, a specific interface protocol, or a mechanism for communication among components

Architectural Design

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design.
- The output of this design process is a description of the software architecture

Architectural Design

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications

Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

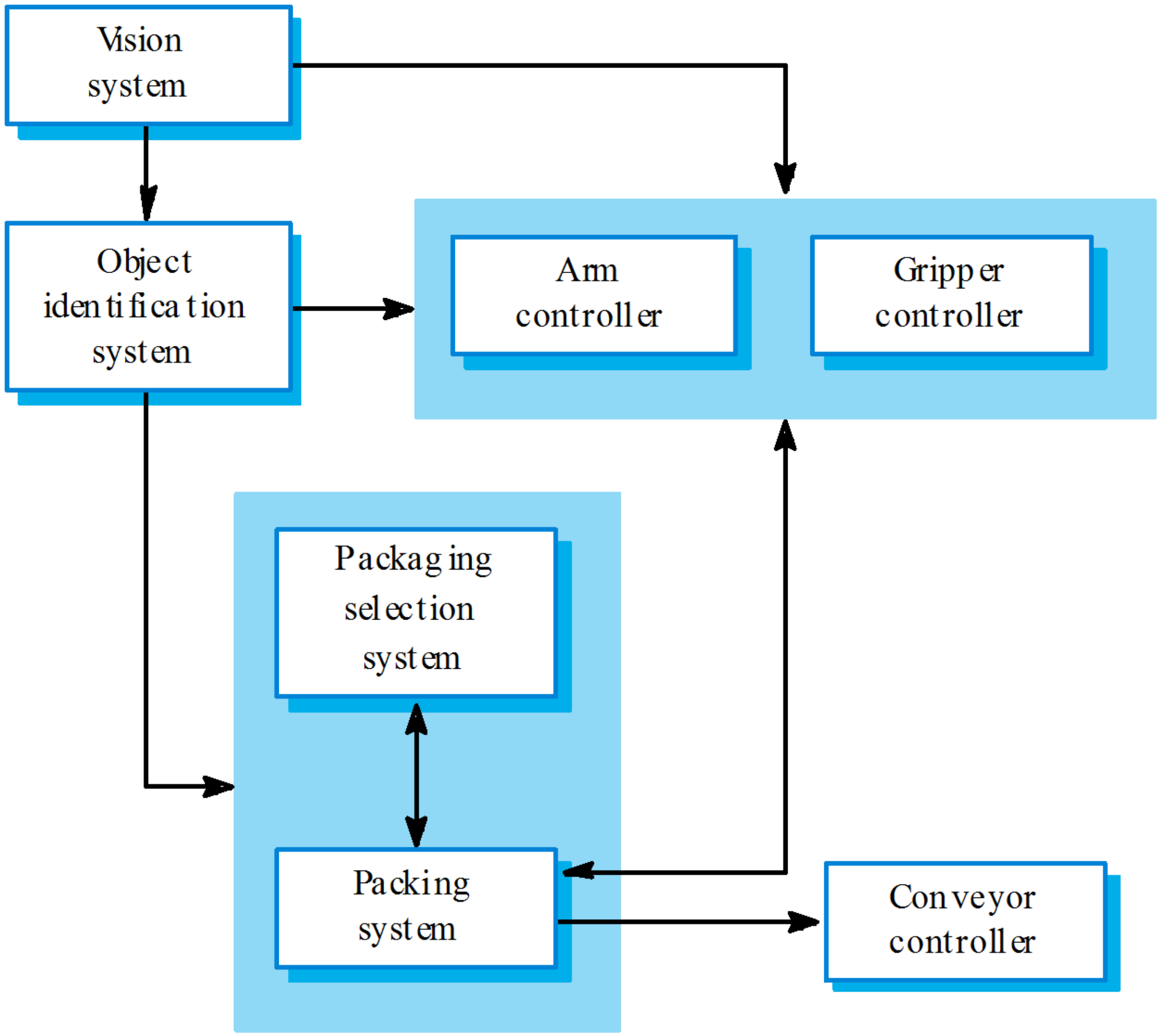
Architecture and system characteristics

- Performance
  - Localise critical operations and minimise communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localise safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components

System structuring

- Concerned with decomposing the system into interacting sub-systems.
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure.
- More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed.

Packing robot control system



Box and line diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

User Interface Design

- User interfaces should be designed to match the skills, experience and expectations of its anticipated users.
- 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.

Human factors in interface design

- Limited short-term memory
  - People can instantaneously remember about 7 items of information. If you present more than this, they are more liable to make mistakes.
- People make mistakes
  - When people make mistakes and systems go wrong, inappropriate alarms and messages can increase stress and hence the likelihood of more mistakes.
- People are different
  - People have a wide range of physical capabilities. Designers should not just design for their own capabilities.
- People have different interaction preferences
  - Some like pictures, some like text.

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.

User interface 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 principles

- Recoverability
  - The system should provide some resilience to user errors and allow the user to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.
- User guidance
  - Some user guidance such as help systems, on-line manuals, etc. should be supplied
- User diversity
  - Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available

Interaction styles

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

Interaction Styles

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement. Only suitable where there is a visual metaphor for tasks and objects.	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users. Can become complex if many menu options.	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn Checkable	Takes up a lot of screen space. Causes problems where user options do not match the form fields.	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn. Poor error management.	Operating systems, Command and control systems
Natural language	Accessible to casual users Easily extended	Requires more typing. Natural language understanding systems are unreliable.	Information retrieval systems