0208504I

BE, IT-2/4

01/3/2009

SE Assignment-II

1) Write about Golden Rules in design of user interface.

Ans:- Golden Rules:-

In interface design, Theo Mandel [MAN97] coins 3 "golden rules". 1. Place user in control

2. Reduce user's memory load

3. Make interface consistent.

1. Place user in control:-

Mandel [MAN97] defines a number of design principles that allow the user to maintain control.

Define interaction modes in a way that does not force a user into unnecessary or undesired actions: An interaction mode is the current state of interface. The user should be able to enter and exit the mode with little or no effort.

Provide flexible interaction: Because different users have different interaction preferences, choices should be provided. Consider, for example, the difficulty of using keyboard commands to draw a complex shape.

Allow user interfaction to be interruptible & undoable. Even when involved in a sequence of actions, user should be able to interrupt the sequence to do some thing else. The user should also be able to "undo" any action.

Streamline interaction as skill levels advance & allow interaction to be customized. Users often find that they perform same sequence of interactions repeatedly. It is worthwhile to design a "macro" mechanism that enables an advanced user to customize interface to facilitate interaction.

Hide technical internals from casual user. The user interface should move user into virtual world of application. The user should not be aware of OS, file management fn.s, or other arcane computing technology. In essence, interface should never require that user interact at a level that is "inside" the machine.

Design for direct interaction with objects that appear on screen. The user feels a sense of control when able to manipulate objects that are necessary to perform task in a manner similar to what would occur if object were a physical thinking

2. Reduce the User's Memory Load :—

Mandel [MAN97] defines design principles that enable an interface to reduce user's memory load. Reduce demand on short-term memory: When users are involved in complex tasks, demand on short-term memory can be significant. The interface should be designed to reduce requirement to remember past actions & results. This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than having to recall them.

Establish meaningful defaults: Initial set of defaults should make sense for average user, but user should be able to specify individual preferences. However, a "reset" option

should be available, enabling redefination of original default values.

Define shortcuts that are intuitive: When mnemonics are used to accomplish a system fn., the mnemonic should be tied to action in a way that is easy to remember.

Visual layout of interface should be based on real world metaphor: This enables user to rely on well-under-stood visual cues, rather than memorizing an arcane interaction sequence.

Disclose information in a progressive fashion: The interface should be organized hierarchally. That is, info about a task, an object, or some behaviour should be presented first at a high level of abstraction. More detail should be presented after user indicates interest with mouse pick.

3) Make The Interface Consistent:-

The interface should present and acquire info in a consistent fashion. Mandel [MAN97] defines a set of design principles that help make interface consistent.

Allow the user to put current task into meaningful context. Many interfaces implement complex layers of interactions with dozens of screen images. It is imp. to provide indicators that enable user to know context of work at hand. In addition, user should be able to determine where he has come from & what alternatives exist for transition to new task.

Maintain consistency across family of applications. A set of applications should all implement same design rules so that consistency is maintained for all interaction.

If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Once a particular interactive sequence has become a de facto standard, the user expects this in every application she encounters. A change will cause confusion.

2) Write about Architectural Styles & Patterns.

Ans:- Architectural Styles & Patterns :-

Architectural style is a transformation that is imposed on design of an entire system.

Architectural pattern imposes a transformation on design of an architecture. However, a pattern differs from style in no. of fundamental ways:

(i) scope of pattern is less broad.

(ii) pattern imposes a rule on architecture.

(iii) architectural patterns tend to address specific behavioral issues within context of architectural.

Architectural Styles:—

Although millions of computer-based systems have been created over past 50 years, the vast majority can be categorized into one of relatively small no. of architectural styles.

Data-centred architecture:— A data store resides at centre

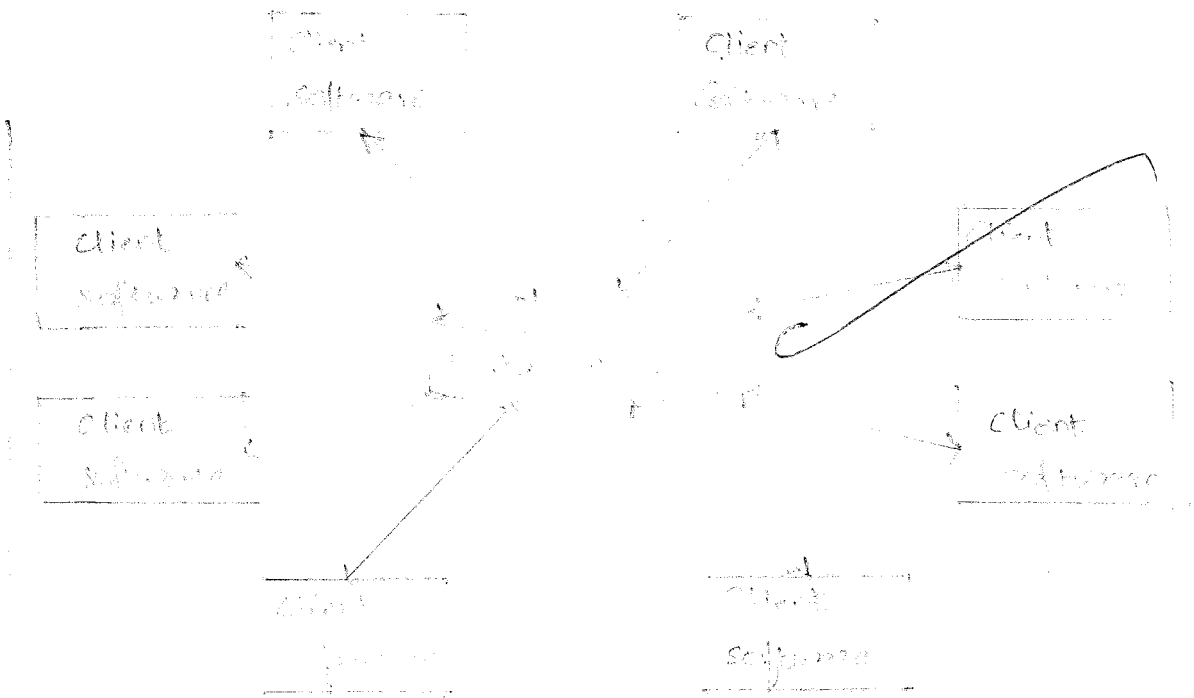


Fig.1

of this architecture & is accessed frequently by other components that update, add, delete or else modify data within store. Fig.1 illustrates typical data centred style.

A data-centred architecture promotes integrability. In addition, data can be passed among clients using blackboard mechanism.

Data-flow Architecture: This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.

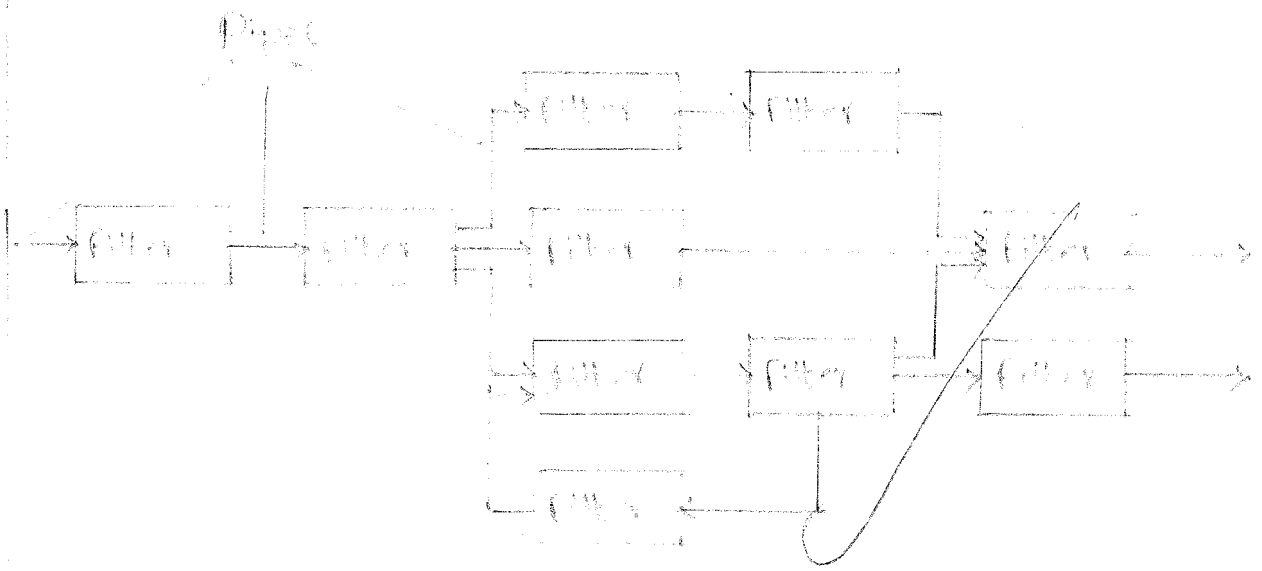


Fig-2

A pipe and filter structure (Fig-2) has set of components called filters, connected by pipes that transmit data from one component to next. Each filter works independently of those components upstream & downstream, is designed to expect data i/p of certain form & produces data o/p of specified form.

Object-oriented Architecture: Components of system encapsulate data & operations that must be applied to manipulate data. Communication & coordination b/w components is accomplished via message passing

Layered Architecture: Basic structure of layered architecture is illustrated in Fig. 3. A no. of diff. layers are defined, each accomplishing operations that progressively

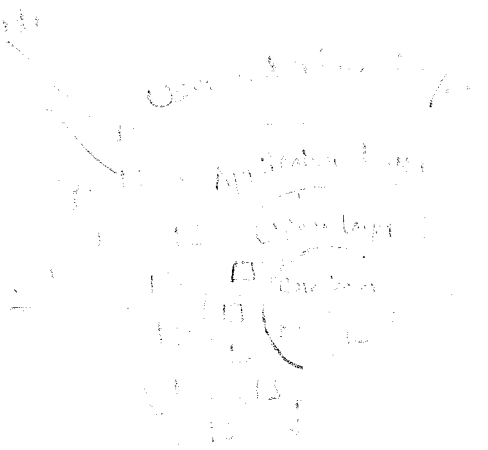


Fig 3

become closer to machine instruction set. At outer layer, components service user interface operations. At inner layer, components perform OS interfacing. Intermediate layers provide utility services & application software functions.

Architectural Patterns:-

As it has already been noted, architectural patterns for s/w define specific approach for handling some behavioral characteristic of system. Bosch [bosco] defines no. of architectural pattern domains. Representative examples are provided in paras that follow.

Concurrency:— Many applications must handle multiple tasks in a manner that simulates parallelism. There are a no. of different ways in which an application can handle concurrency, & each can be presented by different architectural pattern.

Persistance:— Data persists if it survives past the execution of process that created it. Persistent data are stored in database or file & may be read or modified by other process at a later time. In object-oriented environments, idea

of persistent object extends persistence concept a bit further. In general, 2 architectural patterns are used to acheive persistance - a database management system pattern that applies storage & retrieval capability of DBMS to application architecture or an application level persistence pattern that builds persistance features into application architecture.

Distribution:— The Distribution problem addresses manner in which systems or components within systems communicate with one another in distributed environment. There are 2 elements to this problem. (i) the way in which entities connect to one another, (ii) nature of communication that occurs. The most common architectural pattern established to address distribution problem is broker pattern. A broker acts as "middle-man" b/w client component & server component. Client sends a message to broker, & broker completes connection. CORBA is an eg. of broker architecture.

3) Write about component-level Design.

Ans:- Conducting component-level Design;-

The designer must transform information from analysis and architectural models into design representation that provides sufficient detail to guide the construction (coding and testing) activity. The following steps represent a typical task set for component-level design, when it is applied for an object oriented system.

Step 1 :- Identify all design classes that correspond to the problem domain.

Using analysis & architectural models, each analysis class & architectural component is elaborated.

Step 2 :- Identify all design classes that correspond to infrastructure domain.

These classes are not described in analysis model & are often missing from architecture model,

but they must be described at this point.

Step 3:- Elaborate all design classes that are not acquired as reusable components.

- Elaboration requires that all interfaces, attribute & operations necessary to implement the class be described.

- Design heuristics must be considered as this task is conducted.

Step 3a:- Specify message details when classes or components collaborate.

- Showing the details of these collaborations by specifying structure of messages that are passed b/w objects within a system.

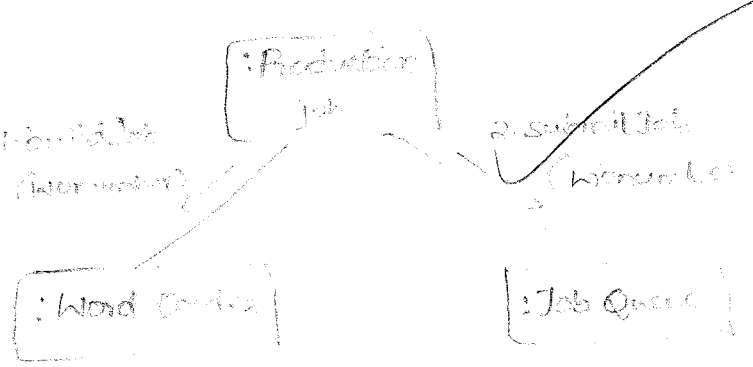
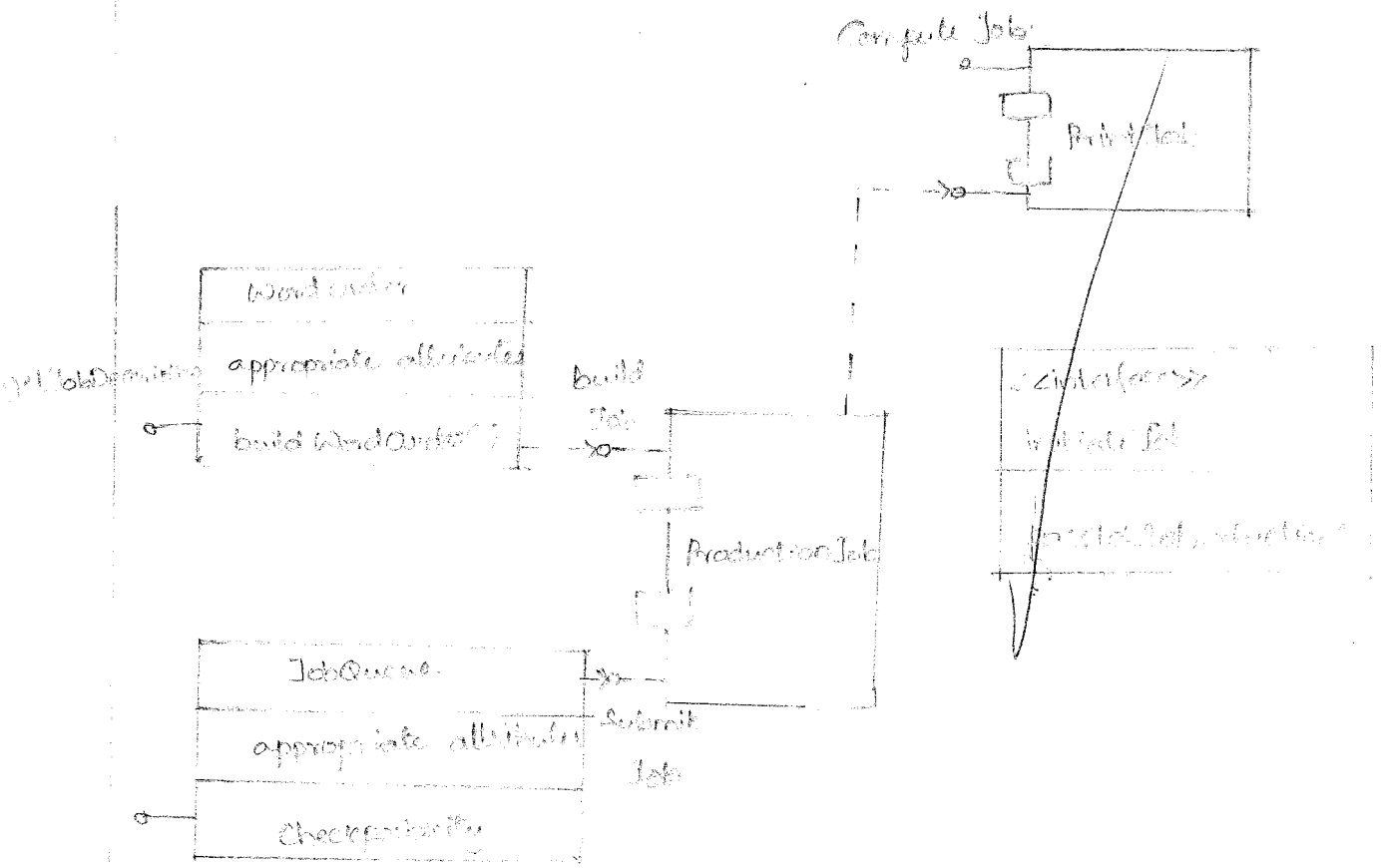


Fig a

Fig.a illustrates simple collaboration with messaging

Step-3b :- Identify appropriate interface for each component.

- UML interface, " a group of externally visible operations. This contains no. of internal structure, it has no attributes, no associations . . . ".



Refactoring interfaces > class definition

for

Step 3c :- Elaborate attributes & define data types & data structures required to implement them

Step 3d :- Describe processing flow with operation in detail.

Step-4 :- Describe persistant data sources (data bases & files) & identify classes required to manage them.

As a part of architectural design, Databases & files normally transcend, the design description of individual component.

Step 5 :- Develop & elaborate behavioral representations for a class or component.

- It is necessary to model the behavior of design class
- UML state diagrams are used to represent the externally observable behavior of system & ~~the~~ more localised behavior of individual analysis classes.

Step-6: Elaborate deployment diagrams to provide additional implementation detail.

• Major system functions are represented within the context of computing environment that will house them.

• Deployment diagrams can be elaborated to represent the location of key packages of components, to avoid diagramatic complexity.

Step-7: factor every component-level design representat-ion & always consider alternatives.

• It is essential to refactor as design work is conducted.

• A design should not suffer from tunnel vision.

• Develop alternatives & consider each carefully using design concepts & principles.

13/3/09