JUNAID . GIRKAR
60004190057
TE COMPS A4

SOFTWARE ENGINEERING
ASSIGNMENT - 1

Q1 Elaborate the task set for creating component level design in OO projects.

ANS

- Component level design is elaborative in nature.
- It transforms information from requirements and architectural models into a 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 applied to object-oriented projects:-

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

→ Using the requirements and architectural model, each analysis class and architectural component is elaborated.

STEP 2 : Identify all design classes that correspond to the infrastructure domain.

→ These classes are not described in the requirements model and are often missing from the architecture model, but they must be described at this point.

→ Classes and components in this category include :

- GUI components.

- operating system components
- object and data management components.

STEP 3: Elaborate all design classes that aren't acquired as reusable components.

→ Elaboration requires that all interfaces, attributes and operations necessary to implement the class be described in detail.

→ Design heuristics [e.g. component cohesion and coupling]

STEP 3(a): Specific message details when classes or components collaborate.

→ The requirements model makes use of a collaboration diagram to show how analysis classes collaborate with one another.

STEP 3(b): Identify appropriate interfaces for each component.

→ within the context of component-level design, a UML interface is a group of public [externally visible] operations.

→ The interface contains no internal structure, has no attributes or associations.

STEP 3(c): Elaborate attributes and define data types and data structures required to implement them.

→ In general, data structures and types used to define attributes are defined within the context of the programming language that is to be used for the implementation.

→ UML defines an attribute's data type using the following

syntax :

name : type-expression initial-value {property string}

→ Here, name is the attribute name, type expression is the data type, initial value is the value that the attribute takes when an object is created, and property-string defines a property or characteristic of the attribute.

STEP 3(D) : Describe processing flow within each operation in detail.

→ This may be accomplished using a programming language-based pseudocode or with a UML activity diagram.

→ Each software component is elaborated through a number of iterations that apply the stepwise refinement concept.

STEP 4 : Describe persistent data sources [databases and files] and identify the classes required to manage them.

→ Databases and files normally transcend the design description of an individual component. In most cases, these persistent data stores are initially specified as part of architectural design.

STEP 5 : Develop and elaborate behavorial representations for a class or component.

→ UML state diagrams were used as part of the requirements model to represent the externally observable behaviour of the system.

STEP 6: Elaborate deployment diagrams to provide additional implementational detail. Deployment diagrams are used as part of architecture.

STEP 7: Refractor every component-level design representation and always consider alternatives.

Q2 Explain the Golden Rules of User Interface Design.

ANS. User Interface Design creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

The Golden Rules of User-Interface Design are :-

(i) Place the user in control

a → Design interaction modes in a way that does not force a user into unecessary or undesired actions.

- An interaction mode is the current state of the interface. For example, if spell check is selected in a word-processor menu, the software moves to a spell-checking mode. The user should be able to enter and exit the mode with little or more effort.

b → Provides flexible interaction

- Because different users have different interaction preferences, choices should be provided

- For example, software might allow a user to interact via keyboard commands, mouse movements, a digitizer pen, a multi-touch screen or voice recognition commands. However, not every action is amenable to every interaction mechanism.
- For example, there will be difficulty in using keyboard commands or voice inputs to draw a complex shape

c → Allow user interaction to be interruptible and undoable
- Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that's already been done).
- The user should also be able to "undo" any action.

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

e → Hide technical internals from the casual user
- The user interface should move the user into the virtual world of the application
- The user should not be aware of the operating system, file management functions, or other arcane computing technology.

- In essence, the interface should never require that the user interact at a level that is "inside" the machine. [Eg: A user should never be required to type os commands from within application software.]

f → Design for direct interaction with objects that appear on the screen.

- The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing.
- For example, an application interface that allow a user to "stretch" an object [scale in size] is an implementation of direct manipulation.

(ii) Reduce the user's memory load.

- The more a user has to remember, the more error-prone the interaction with the system will be.
- It is for this reason that a well-designed user interface doesn't tax the user's memory.
- whenever possible, the system should "remember" pertinent information and assist the user with an interaction scenario that the assist recalls.

a → Reduce demand on short-term memory.

- when users are involved in complex tasks, the demand on short-term memory can be significant.
- The interface should be designed to reduce the requirement to remember past actions, inputs and results.

- This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than recalling them.

b → Establish meaningful defaults
- The initial set of defaults should make sense to the average user, but he should be able to specify his individual preferences as well.
- However, a "reset" option should be available, enabling the redefination of original 'default values.

c → Define shortcuts that are intuitive
- When mneumonics are used to accomplish a system function [e.g: alt + P to invoke the print function], the mneumonic should be tied to the action in a way that is easy to remember. [e.g. First letter of the task invoked].

d → The visual layout of the interface should be based on a real-world metaphor
- For example, a bill payment system should use a checkout and check register metaphor to guide the user through the bill paying process.
- This enables the user to rely on well-understood visual cues rather than memorizing an arcane interaction sequence.

e → Disclose information in a progressive fashion.
- The interface should be organized hierarchically. Information about a task, an object, or some

behaviour should be presented first at a higher level of abstraction with details following when user indicates interest.

- For example, in word processing applications, underlining function is available in the style menu. When the user picks it, all underlying options, such as single underline, double underline, dashed underline are presented.

(iii) Make the interface consistent.

- The interface should present and acquire information in a consistent fashion. This implies that

(1) All visual information is organized according to design rules that are maintained throughout all screen displays.

(2) Input mechanisms are constrained to a limited set that is used consistently throughout the application.

(3) Mechanisms for navigating from task to task are consistently defined and implemented.

Design principles :-

a → Allow the user to put the current task into a meaningful content.

- Many interfaces implement complex layers of interactions with dozens of screen images.
- It is important to provide indicators [eg. window titles, graphical icons, consistent color coding], that enables the user to know the context of the work at hand.

- In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

b → Maintain consistency across a family of applications.

- A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interactions.

c → If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

- For example, once a particular interactive sequence has become a defacto standard [e.g: use of ctrl + V for pasting], the user expects this in every application he encounters. A change [eg. using alt+V to invoke pasting] will cause confusion.