Foundation for Advancement of Science & Technology

Advantages

→ Freedom & Adaption
→ High-quality, low-risk product
→ Reduces development time upto 40%
→ Scrum customer satisfaction is very important
→ Reviewing the current sprint before moving to new one.
→ Manageable and understandable chunks.
→ Complete visibility → ↑ team communication
→ Trust b/w customer and developers.

Disadvantages

↳ More efficient for small team size.
↳ No changes in the sprint.

* Architectural Design :- is a process for identifying the sub-systems making up a system and the framework for sub-system control and communication.
-) Output of this design → software architecture
-) Early stage of system design process.
-) Link between specification and design processes (often carried out in parallel).
-) Involves identifying major system components and their communication.

Software architecture can be designed at two level of abstraction
-) Architecture in small is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
-) Architecture in large is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are

distributed over different computers, which may be owned and managed by different companies.

- Advantages

-) Stakeholder communication - Architecture may be used as a focus of discussion by system stakeholders.
-) System analysis - Well-documented architecture enables the analysis of whether the system can meet its non-functional requirements.
-) Large scale reuse - The architecture may be reusable across a range of systems or entire line of products.

* Architectural representation (Box and line diagrams)

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

* Architecture Design Processing :-

-) ...
-) Analyze requirements
-) Define business use case (functional / non-functional)
-) Choose architecture pattern / style.
-) Communicate architecture to stakeholders (prototypes)
-) Evaluate architecture
-) Design detailed implementation architecture
-) ...

* Uses of Architectural Models :-
-) facilitating discussion about the system design.
-) Documenting an architecture that has been designed.

* Agility and Architecture Design
-) early stage of agile processes is to design an overall systems architecture
-) Architecture design activities takes place throughout the project.

* Architecture Reuse :-
-) Systems in the same domain often have similar architectures that reflect domain concepts.
-) Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
-) Architecture of a system may be designed around one of more architectural patterns or styles.

* 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.

Foundation for Advancement of Science & Technology

* Architectural Views :-

→ Shows one view or perspective of the system.
→ how system is decomposed into modules, run-time procen interact, ways components are distributed across a network.

a) 4+1 model :-

→ A logical view, which shows the key abstractions in the system as object or object classes.
→ A process view, which shows how, at run-time, the system is composed of interacting processes.
→ A development view, which shows how the software is decomposed for development.
→ A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
→ Related using use cases or scenarios (+1)

*) Architectural Patterns :-

→ ~~mean~~ pattern means of representing, sharing and reusing knowledge
→ Architectural pattern is a stylized description of a good design practice
→ pattern should include when they are useful or not.
→ pattern may be represented using tabular and graphical descriptions

* Model-View-Controller :-

→ Serves as a basis of interaction management in many web-based systems.
→ components :-

→ The model is the central component of the pattern that directly manage the data, logic and rules of the application. Independant of user-interface.
→ A view can be any output representation of information, such as chart or diagram. Multiple views of same info are possible.
→ A controller accepts input and converts it to commands for the model of view.

•) Problem :-
→ display presented frequently changes in response to input or computation
→ Different needs how to view the program's information
→ need reflect data changes to all users in a way they want information
→ While making it easy to make changes to UI.

•) Solutions :-
→ Seperating the data and displaying, using three components → model → view → controller.

→ Advantages :-
•) views and controllers can be easily added, removed or changed
•) views can be added or changed during execution.
•) UI components can be changed, even at run-time.

→ Disadvantages :-
•) Views and controllers are often hard to seperate
•) Frequent updates may slow data display and degrade UI performance
•) UI components (view, controller) highly dependent on model components.

Foundation for Advancement of Science & Technology

* Layered Architecture :-

- Used for interfacing of sub-systems
- organizes the system into a set of layers each provide set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artifical to structure systems in this way.

- When Used? → building new facilities on top of existing systems.
→ Development is spread across several teams with responsibility of a layer.
→ Requirement for multi-level security.

- Advantages :-
→ replacement of entire layers as long as interface is maintained.
→ Redundant facilities (eg authentication) can be provided to increase the dependability of the system.

- Disadvantages :-
→ High level layer may have to interact with lower level layer rather than layer immediately below it.
→ Performance can be a problem as it is processed at each layer because of multiple levels of interpretation of a service request.

* Repository Architecture :-

All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.

Foundation for Advancement of Science & Technology

- When Used ?
→ System where large volumes of information are generated that has to be stored for a long time.
→ Data-Driven systems where inclusion of data in the repository triggers an action or tool.

- Advantages :-
→ Independent components.
→ Changes in one component can be propagated to all components.
→ Data can be managed consistently (backups) as it is all in one place.

- Disadvantages :-
→ Repository is a single point of failure
→ Inefficiency in organizing all communications through the repository.
→ Distributing repository across several computers may be difficult.

* Client-Server Architecture :-
→ System is organized into services, with each service delivered from a seperate server. Clients are users of these services and access servers.

- When Used ?
→ Data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also used when the load on a system is variable.

- Advantages :-
→ Can be distributed across a network.

→ General functionality (e.g. printing service) can be available to all clients and doesn't need to be implemented by all services.

•) Disadvantages:-
→ Susceptible to DoS attack or server failure.
→ Unpredictable performance as it depends on the network as well as the system.
→ Management problems if owned by different organizations.

---

* User-Interface Design:-
•) Golden Rules
1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

- Place the user in control:-
•) Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
•) Provide for flexible interaction
•) Allow user interaction to be interruptible and undoable.
•) Hide technical internals from the casual user.
•) Design for direct interaction with objects that appear on the screen.
•) Streamline interaction as skill levels advance and allow the interaction to be customized.

- Reduce the User's load :-
  - Reduce demand on short-term memory.
  - Define shortcuts that are intuitive
  - The visual layout of the interface should be based on a real-world metaphor.
  - Disclose information in a progressive fashion.

- Make the interface consistent :-
  1. Allow the user to put the current task into a meaningful content
  2. Maintain consistency across a family of application
  3. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

* User Interface Analysis and Design :-
- Creation of different models of system function for designing and analyzing.

The user model - establishes the profile of end users of the system.
The design model - A design realization of the user model.
The user's mental model - (system perception) is the image of the system that end users carry in their heads.
The implementation model - combines the outward manifestation of the computer based system (the look and the feel of the interface), coupled with all supporting information (books, manuals, videotapes, help files) that describes interface syntax and semantics.

* The Process :-

→ It is a iterative process and can be represented using a spiral model :-,

① Interface analysis and modelling.
② Interface design,
③ Interface construction
④ Interface validation

Interface analysis focuses on the profile of the users who will interact with the system.

The goal of interface design is to define a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system.

Interface ~~communication~~ construction normally begins with the creation of a prototype that enables usage scenarios to be evaluated. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface.

Interface validation focuses on

- the ability of the interface to implement every user task correctly, to accomodate all task variations, and to achieve all general user requirements
- the degree to which the interface is easy to use and learn, and
- the user's acceptance of the interface as a useful tool in their work.

* Interface Design Principles & Guidelines :-

- Anticipation → anticipate the user's next move
- Communication → communicate the status of any activity initiated by user.
- Consistency → navigation controls, menus, icons and aesthetics (e.g. color, shape, layout) should be consistent.

- Controlled autonomy → facilitate user movement that enforces navigation conventions.
- Efficiency → Should optimize user's work efficiency, not developers or client-server environment
- Flexibility → accomplish task directly and allows to explore.
- Focus → focused on the user task(s) at hand.
- Fitt's Law → is an effective method of modelling rapid, aimed movements, where one appendage (like a hand) starts at rest at a specific start position and moves to rest within a target area.
- Human Interface Objects → Any interface object that can be "seen, heard, touched or otherwise perceived."
- Latency Reduction → Rather than making user wait for internal operations, multitasking in a way that lets user proceed with operation.
- Learnability → minimize learning time.

---

* Design concepts :-

→ To generate model which shows firmness, delight and commodity

* Software Quality Guidelines :-

→ Recognizable architectural style and compose a good design.
→ Must be Modular, i.e logically partitioned into elements.
→ representation of data, architecture, interface and components should be distinct.
→ Appropriate data structures and recognizable data patterns.
→ Design components must show the independent functional characteristics.
→ Creates an interface that reduces the complexity.
→ A design must be derived using the repeatable method.
→ Notations for effective communication.

Foundation for Advancement of Science & Technology

* Quality Attributes :- FURPS

- Functionality → evaluates the feature set and capabilities of program
- Usability → It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.
- Reliability → It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, recovery from failure.
- Performance → It is measured by considered processing speed, response time, resource consumption, throughput and efficiency.
- Supportability → maintainability, compatibility, modularity, reusability

* Design Concepts Fundamentals :-

1. Abstraction →
- At highest level, solution is stated in large terms
- Lower level provides a more detail description of solution.
- Sequence of instruction that contains function refers in a procedural abstraction.
- Collection of data that describes a data object is a data abstraction

2. Architecture →
- Complete structure of software is known as software architecture.
- Structure provides conceptual integrity.
- Architecture is the structure of program modules where they interact with each other in a specialized way.

3. Patterns →

A design pattern describes a design structure that structure solves a particular design problem in a specified content.

4. Modularity →

- Seperately divided components which integrate to satisfy the problem requirements
- Permits a program to be managed easily.

5. Information Hiding →

- Data presented in a module is not accessible for other module not requiring that information

6. Functional Independence →

- Concept of seperation and related to modularity, abstraction and information hiding. The functional independence is accessed using two criteria i.e Cohesion and coupling.

→ Cohesion :

- Extension of information hiding concept
- Cohesive module performs a single task and it requires a small interaction with the other components in other parts of program

→ Coupling :

- Indication of interconnection between modules in a structure of software.

7. Refinement →

- Top-down design approach
- Process of elaboration
- A program is established for refining levels of procedural details.
- Hierarchy is established by decomposing a statement of function.

Foundation for Advancement of Science & Technology

8. Refactoring →

- Reorganization technique which simplifies the design of components without changing its function behaviour.
- Doesn't change external behaviour, improves internal structure.

9. Design Classes →

- Software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on the features of the problem which are user visible.

Five different types of design classes :-

1) User Interface class
- These classes are designed for Human computer Interaction (HCI)
- Defines all abstraction which is required for HCI.

HCI: Design, development and evaluation of computer systems that are usable, efficient and satisfying for people to interact with.

2) Business Domain Classes :-
- Refinements of analysis classes
- Recognized as Attributes and methods which are required to implement the elements of the business domain.

3) Process classes :-
- Implements the lower level business abstraction.

4) Persistance classes :-
- It shows data stores that will persist behind the execution of the Software.

5) System Class :-
Implements software management and control functions that allow to operate and communicate in computing environment and outside world.

* Design Class characteristics :-
1. Complete and sufficient
-) Must be total encapsulation of all attributes and methods.

2- Primitiveness
-) Method in the design class should fulfill one service for the class.
-) If service is implemented with a method then the class should not provide another way to fulfill same thing.

3- High cohesion :-
-) A cohesion design class has a small and focused set of responsibilities

4. Low coupling :
-) All the design classes should collaborate with each other in a design model.
-) Minimum acceptable of collaboration must be kept on this model.
-) If a design module is highly coupled then the system is difficult to implement, test and maintain.

* Types of Design Elements :-
1. Data Design Elements :-
-) Represents high level of abstraction
-) Then it is more refined which is processed by the computer based system

- Structure of data is most important part of the software design.

2. Architectural Design Elements :-
- Provides us overall view of the system
- Represented as a set of interconnected subsystem that are derived from analysis package in the requirement model.

Architectural model is derived from
- information
  - model elements
  - architectural style and patterns.

3. Interface Design Elements :-
- Information flow within it and out of the system.
- Communicate between the components as part of architecture.

Important elements of the interface design.
1. The user interface
2. The external interface to the other systems, networks etc.
3. The internal interface between various components.

4. Component level Diagram Elements :-
- Internal details of the each software component.
- Processing of data structures occurs in a component and an interface which allows all the component operations.

5) Deployment level Design Elements :-
- Shows the functionality and subsystems that allocated in the physical computing environment which support the software.
- Three computing environment : Personal computer, EPI server and Control Panel.

Foundation for Advancement of Science & Technology

System Modeling

System modelling is the process of developing abstract models of system, with each model presenting a different view or perspective of that system. Representing using graphical notation, now likely in UML.
Models helps the analyst to understand the functionality of the system.
↳ they are used to communicate with customers.
Models can explain the system from different perspective.

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions b/w a system and its environment, or b/w the components of a system
- A structural perspective where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behaviour of the system and how it responds to events.

Five Types of UML Diagrams :-

- Activity diagrams, which show the activities involved in a process.
- Use case diagrams, which shows the interaction b/w system and environment
- Sequence diagrams, " " " " b/w actors and the system and between system components.
- Class diagrams, shows object classes and association b/w these classes.
- State diagrams, shows how system reacts to internal and external events.

* Context and Process models :-

Context model are used to illustrate the operational context of a system. They show what lies outside the system boundaries.

System boundaries are established to define what is inside and what is outside the system.
context models show the other systems in the environment, not how the system being developed is used in that environment.
procen models reveal how the system being developed is used in that environment. UML may be used to define business procen models.

* Interaction Models :-
types of interaction that can be presented in a model
- Modelling user interaction is important as it helps to identify user requirements.
- Modelling system-to-system interaction highlights the communication problems that may arise.
- Modelling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Use case can be represented using UML.

UML sequence diagrams are used to model the interaction b/w the actors and the objects within a system.

* Structural Models :- → Static → Show the structure of the system design
→ Dynamic → Show the organization of system when executing.

Display the organization of a system in terms of the components that make-up that system and their relationships.

UML class diagrams are used when developing an object-oriented system model to show the classes in a system and its associations

Generalization is used to manage complexity. It is often useful to examine the classes in a system

* Behavioral Model :-
They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
Two types of stimuli
1) Some data arrives that has to be processed by the system.
2) Some event happens that triggers system processing. Events may have associated data.

Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
Event-Driven models shows how a system responds to external and internal events.