Design Engineering

❖ Design Process & quality

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels or phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design

Elements of a System

- Architecture: This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
- Modules: These are components that handle one specific task in a system. A combination of the modules makes up the system.
- Components: This provides a particular function or group of related functions. They are made up of modules.
- Interfaces: This is the shared boundary across which the components of a system exchange information and relate.
- Data: This is the management of the information and data flow.

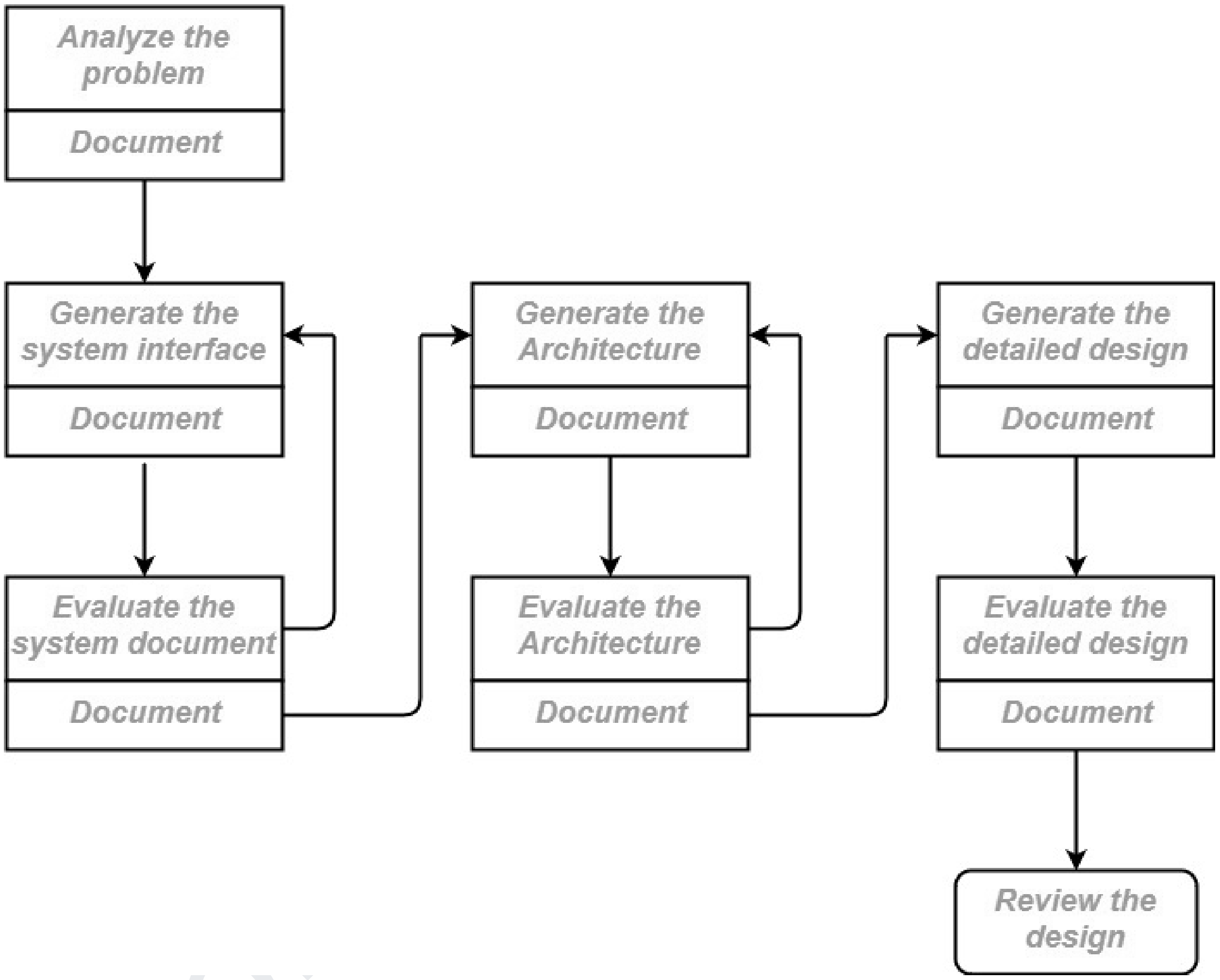

Analyze the problem
Document
Generate the system interface
Document
Evaluate the system document
Document
Generate the Architecture
Document
Evaluate the Architecture
Document
Generate the detailed design
Document
Evaluate the detailed design
Document
Review the design

Figure: Software Design Process

Interface Design

Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored, and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents.

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification of the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces.
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design

Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces.
- Unit states and state changes.
- Data and control interaction between units.
- Data packaging and implementation, including issues of scope and visibility of program elements.
- Algorithms and data structures.

Elements of the Requirements Model

Requirements for a computer-based system can be seen in many different ways. Some software people argue that it's worth using a number of different modes of representation while others believe that it's best to select one mode of representation. The specific elements of the requirements model are dedicated to the analysis modeling method that is to be used.

- Scenario-based elements : Using a scenario-based approach, the system is described from the user's point of view. For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases. Figure 1(a) depicts a UML activity diagram for eliciting requirements and representing them using use cases. There are three levels of elaboration.

- Class-based elements : A collection of things that have similar attributes and common behaviors i.e., objects are categorized into classes. For example, a UML case diagram can be used to depict a Sensor class for the SafeHome security function. Note that the diagram lists attributes of sensors and operations that can be applied to modify these attributes. In addition to class diagrams, other analysis modeling elements depict the manner in which classes collaborate with one another and relationships and interactions between classes.

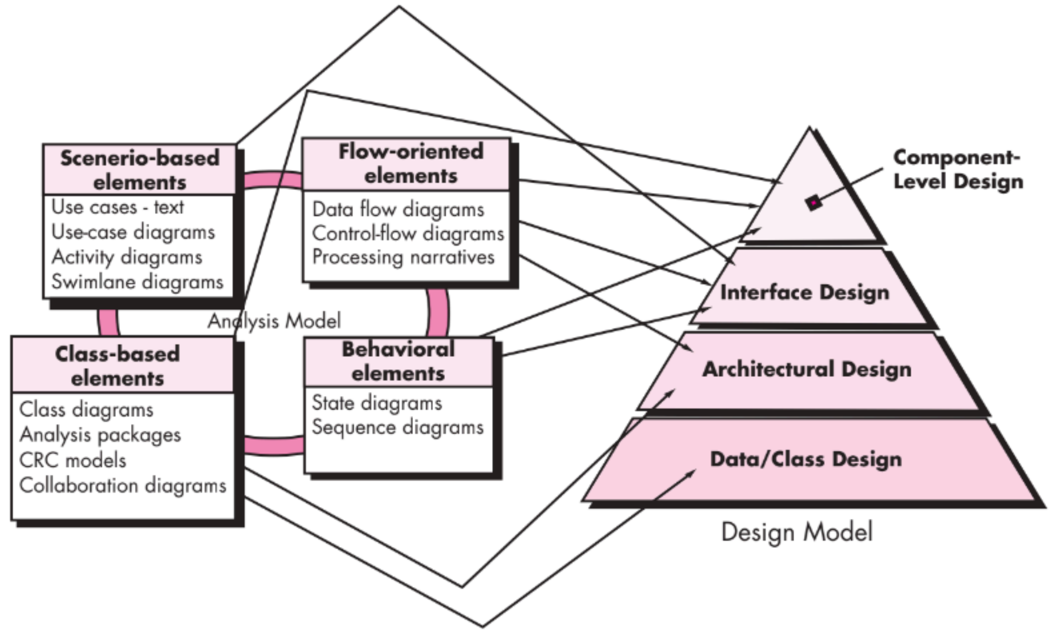
- Behavioral elements : Effect of behavior of computer-based systems can be seen on design that is chosen and implementation approach that is applied. Modeling elements that depict behavior must be provided by the requirements model.

- Method for representing behavior of a system by depicting its states and events that cause the system to change state is a state diagram. A state is an externally observable mode of behavior. In addition, a state diagram indicates actions taken as a consequence of a particular event. To illustrate use of a state diagram, consider software embedded within the safeHome control panel that is responsible for reading user input. A simplified UML state diagram is shown in figure 2.

- Flow-oriented elements : As it flows through a computer-based system information is transformed. System accepts input, applies functions to transform it, and produces output in various forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. Transform may compromise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.

Output produces a 200-page report or may light a single LED. In effect, we can create a flow model for any computer-based system, regardless of size and complexity.

FIGURE 8.1 Translating the requirements model into the design model



Each of the elements of the requirements model provides information that is necessary to create the four design models required for a complete specification of design. The flow of information during software design is illustrated in Figure 8.1. The requirements model, manifested by scenario-based, class-based, flow-oriented, and behavioral elements, feed the design task. Using design notation and design methods discussed in later chapters, design produces a data/class design, an architectural design, an interface design, and a component design.

The data/class design transforms class models (Chapter 6) into design class realizations and the requisite data structures required to implement the software.

The objects and relationships defined in the CRC diagram and the detailed data content depicted by class attributes and other notation provide the basis for the data design action. Part of class design may occur in conjunction with the design of software architecture. More detailed class design occurs as each software component is designed.

The importance of software design can be stated with a single word—quality. Design is the place where quality is fostered in software engineering. Design provides you with representations of software that can be assessed for quality. Design is the only way that you can accurately translate stakeholder's requirements into a finished software product or system. Software design serves as the foundation for all the software engineering and software support activities that follow. Without design, you risk building an unstable system—one that will fail when small changes are made; one that may be difficult to test; one whose quality cannot be assessed until late in the software process, when time is short and many dollars have already been spent.

Introduction to design process

Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software. Initially, the blueprint depicts a holistic view of software. That is, the design is represented at a high level of abstraction—a level that can be directly traced to the specific system objective and more detailed data, functional, and behavioral requirements. As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction.

These can still be traced to requirements, but the connection is more subtle.

Software Quality Guidelines and Attributes

Throughout the design process, the quality of the evolving design is assessed with a series of technical reviews suggests three characteristics that serve as a guide for the evaluation of a good design:

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

Each of these characteristics is actually a goal of the design process.

<u>Quality Guidelines</u>

In order to evaluate the quality of a design representation, you and other members of the software team must establish technical criteria for good design. For the time being, consider the following guidelines:

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics, and (3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
- A design should contain distinct representations of data, architecture, interfaces, and components.

- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

These design guidelines are not achieved by chance. They are achieved through the application of fundamental design principles, systematic methodology, and thorough review.

Quality Attributes

Hewlett-Packard [Gra87] developed a set of software quality attributes that has been given the acronym FURPS—functionality, usability, reliability, performance, and supportability. The FURPS quality attributes represent a target for all software design:

- Functionality is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.
- Usability is assessed by considering human factors (Chapter 11), overall aesthetics, consistency, and documentation.

- Reliability is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.
- Performance is measured by considering processing speed, response time, resource consumption, throughput, and efficiency.
- Supportability combines the ability to extend the program (extensibility), adaptability, serviceability—these three attributes represent a more common term, maintainability—and in addition, testability, compatibility, configurability, the ease with which a system can be installed, and the ease with which problems can be localized.

❖ Design Concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detailed description of the solution.
- A sequence of instructions that contain a specific and limited function refers to a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.

- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

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

4. Modularity

- A software is separately divided into name and addressable components. Sometimes they are called modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- Functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

- Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statements are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behavior.
- Refactoring is the process of changing the software system in a way that does not change the external behavior of the code and still improves its internal structure.

9. Design classes

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