Design pattern

- A pattern is a named description of a problem and solution that can be applied to new context.
- A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

- The intent of each design pattern is to provide a description that enables a designer to determine:

1. whether the pattern is applicable to the current work.
2. whether the pattern can be reused (hence saving design time)
3. whether the pattern can serve as guide for developing a similar but functionally or structurally different pattern

Component level design

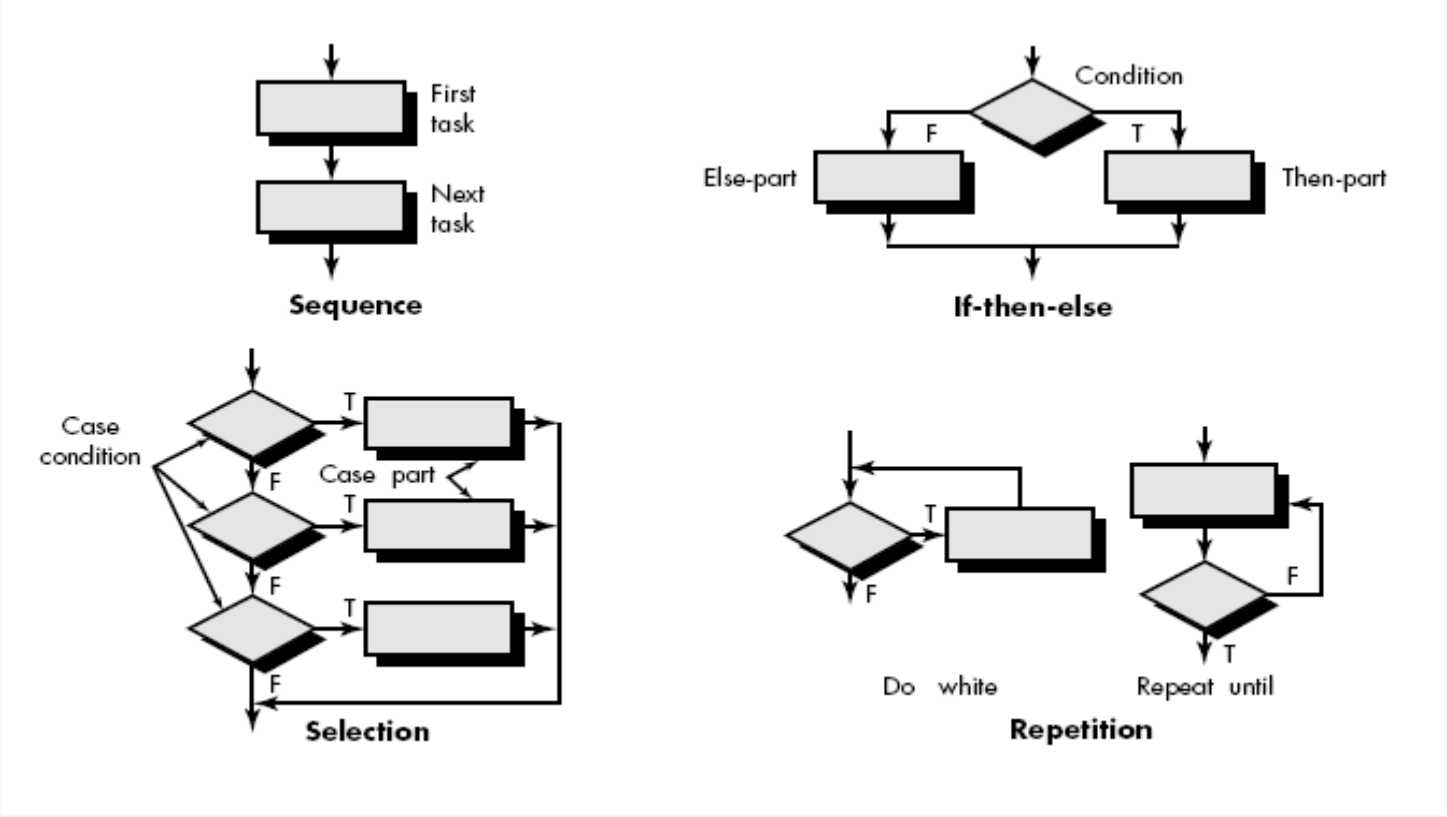
- Component-level design, also called procedural design, occurs after data, architectural, and interface designs have been established. The intent is to translate the design model into operational software. But the level of abstraction of the existing design model is relatively high, and the abstraction level of the operational program is low. The translation can be challenging, opening the door to the introduction of subtle errors that are difficult to find and correct in later stages of the software process.

- It is possible to represent the component-level design using a programming language. In essence, the program is created using the design model as a guide. An alternative approach is to represent the procedural design using some intermediate (e.g., graphical, tabular, or text-based) representation that can be translated easily into source code. Regardless of the mechanism that is used to represent the componentlevel design, the data structures, interfaces, and algorithms defined should conform to a variety of well-established procedural design guidelines that help us to

- **STRUCTURED PROGRAMMING**
- In the late 1960s, Dijkstra and others proposed the use of a set of constrained logical constructs from which any program could be formed. The constructs emphasized "maintenance of functional domain." That is, each construct had a predictable logical structure, was entered at the top and exited at the bottom, enabling a reader to follow procedural flow more easily.
- The constructs are sequence, condition, and repetition. Sequence implements processing steps that are essential in the specification of any algorithm. Condition provides the facility for selected processing based on some logical

Graphical Design Notation

- "A picture is worth a thousand words," but it's rather important to know which picture and which 1000 words. There is no question that graphical tools, such as the flowchart or box diagram, provide useful pictorial patterns that readily depict procedural detail. However, if graphical tools are misused, the wrong picture may lead to the wrong software



- A flowchart is quite simple pictorially. A box is used to indicate a processing step. A diamond represents a logical condition, and arrows show the flow of control. Figure illustrates three structured constructs. The sequence is represented as two processing boxes connected by an line (arrow) of control. Condition, also called ifthen- else, is depicted as a decision diamond that if true, causes then-part processing to occur, and if false, invokes else-part processing. Repetition is represented using two slightly different forms. The do while tests a condition and executes a loop task

- **Tabular Design Notation**
- In many software applications, a module may be required to evaluate a complex combination of conditions and select appropriate actions based on these conditions. Decision tables provide a notation that translates actions and conditions (described in a processing narrative) into a tabular form. The table is difficult to misinterpret and may even be used as a machine readable input to a table driven algorithm.
- Some old software tools and techniques mesh well with new tools and techniques of software engineering. Decision tables are an excellent example. Decision tables preceded software

	Rules								
Conditions	**1**	**2**	**3**	**4**					n
Condition #1	✓			✓	✓				
Condition #2		✓		✓					
Condition #3			✓		✓				
Actions									
Action #1	✓			✓	✓				
Action #2		✓		✓					
Action #3			✓						
Action #4			✓	✓	✓				
Action #5	✓	✓			✓				

- The following steps are applied to develop a decision table:
- 1. List all actions that can be associated with a specific procedure (or module).
- 2. List all conditions (or decisions made) during execution of the procedure.
- 3. Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.
- 4. Define rules by indicating what action(s) occurs for a set of conditions.

- Figure below illustrates a decision table representation of the preceding narrative. Each of the five rules indicates one of five viable conditions (i.e., a T (true) in both fixed rate and variable rate account makes no sense in the context of this procedure; therefore, this condition is omitted). As a general rule, the decision table can be effectively used to supplement other procedural design notation.

	Rules				
Conditions	**1**	**2**	**3**	**4**	**5**
Fixed rate acct.	T	T	F	F	F
Variable rate acct.	F	F	T	T	F
Consumption <100 kwh	T	F	T	F	
Consumption ≥100 kwh	F	T	F	T	
Actions					
Min. monthly charge	✓				
Schedule A billing		✓	✓		
Schedule B billing				✓	
Other treatment					✓