CHAPTER-2

THE DESIGN PATTERNS

2.1 OVERVIEW OF DESIGN PATTERNS
With the growing sophistication and dimension of the computer software systems, knowing and altering of these techniques become challenging jobs, especially when the design and structures documentations are incomplete, missing over period, and sporadic with the source signal. Regaining the first style tradeoffs and choices might assist designers to make change more readily and comprehend big methods. Design designs typically record the design choices and tradeoffs too as potential means for potential evolutions. Therefore, regaining the look designs employed in a pc software program may help to handle the sophistication of big systems. Such retrieval procedures are commonly not completed from scratch but consider benefits of some present reverse engineering resources to remove the significant advice from source signal in to some advanced representations, such as UML diagrams. The function info about its individuals is normally misplaced, whenever a style routine is employed in a layout, however. Regaining such advice in the UML diagrams may assist the developers to talk to additional developers and understand the look. It's difficult to immediately control them, as the UML images are generally saved in certain amazing format. Because the advanced representation to resolve this difficulty, the XMI standards to serialize UML in to XML document can be used. XMI is an XML-based standard suggested from the Object Management Team that maps UML to XML. We use XMI as the advanced representation on the basis of the next factors. First, XMI is an interchange structure for metadata when it comes to the Meta Item Facility. While UML versions are usually endured in some amazing structure of specific application programs, XMI specifies how UML models are planned in to a platform-independent XML document. By symbolizing a UML model in XML, the UML model could be looked for designs.

Minute, a few resources are now accessible to regain the UML versions of the software program from its source signal. There exists plug-ins for these resources that will help get the XMI representations in the UML versions. Therefore, style patterns and the computer software methods modelled in UML diagrams may be mechanically changed into XML document in the XMI structure by this plug - ins. Next, pursuing
the XMI standard enables this design discovery techniques to become normally incorporated with additional techniques and resources following the XMI standard. Fig. 2.1 exhibits an entire structures of the strategy. This style pattern recuperation procedures comprise structural, behavioral, and semantic studies. The system design models are parsed by the structural analysis in XMI to assemble a rectangular matrix whose number of number and lines of posts both equivalent to the number of courses in the supply program. A class is represented by each row, therefore does each line. Each cell of the matrix encodes the associations between the class on the line and the course on the strip. In special, perfect numbers are utilized to encode the associations. Each class-to-class relationship is designated a distinctive perfect number. The worth of every mobile of the matrix is the item of the perfect figures that signify the related associations between the class on the line and the course on the strip. All associations between two courses could be readily decoded from their related matrix mobile value, since the merchandise of prime numbers signifies a distinctive blend of the perfect numbers. Besides coding the class-to-class relationships to the matrix, this strategy encodes the data of every person class into class fat in an identical method. The process or feature of the course is designated a distinctive perfect number. The fat of each course is secured as the item of the perfect numbers to the energy of the amounts of the characteristics, approaches, and associations the class has. This method identifies the style designs making use of an XML document, which comprise their structural, behavioral, and semantic features. These design features are utilized in various periods. Throughout as the system is encoded by us structural evaluation stage, this device extracts the structural details of the design and encodes it in to a dumbbells and matrix in an identical method.

Therefore, the architectural evaluation could be decreased to the coordinating of the weights of the design pattern courses in addition to the style pattern matrix with the program matrix with the weights of the system courses. We phone it a complement so long as there’s a sub matrix of the program matrix such that all cells of the sub matrix are the integrated multiples of the related cells in the style pattern matrix and that the weights of the courses as proven in Fig. 2.1 in the sub matrix are the integral multiples of the weights of the corresponding design pattern courses. The requirements of architectural evaluation could be calm to lessen the false negative cases. As nevertheless, the amount of false positive instances might grow, a result. The false positive instances in the structural evaluation outcomes could be removed in
the later analysis procedures, i.e., the behavior and semantic evaluation. This evaluation checks whether a desirable procedure invocation exists in a course using the appropriate signatures and polymorphic definitions. Various style designs might demand different behavioural studies which could be established from the pattern behavioural features explained in the XML document of pattern description. Some style designs, including Technique and Link, are comparable in their buildings and behaviors. They might simply vary from their motives and intents. The labeling conference of courses could possibly supply some hint of the first style intents and motives, even though no such semantic information is generally retained by source code from program design. For instance, several course titles in the Java.awt comprise "Strategy," that is a great indicator of the initial intents. The naming conventions are checked by our semantic analysis when the variations are essential. As proven in our tests they really help recognize designs in several instances, even though naming conventions aren't consistently discovered by designers. The semantic features that require to assess for every style routine will also be supplied from the design description document in XML. The semantic and behavioral analyses might need examining the source code straight, along with the advanced representations. Nevertheless, such inspections are in line with the outcomes from architectural evaluation to ensure that only special courses and procedures, rather than the whole source signal, are examined.

2.2 FORMAL SPECIFICATION OF DESIGN PATTERNS: STRUCTURAL ANALYSIS

In the preceding area the chief thought of Layout Styles is launched. The design recovery strategy is officially given within this part, to become obvious, more exact, and unambiguous. As our approach to be illustrated by examples, this method utilizes three designs, specifically, the Adapter, Technique, and Composite designs. The architectural analysis focuses on the courses in a computer software program and their characteristics, procedures, and associations with other courses. More especially, we determine set ELM for procedures and characteristics as nicely as set REL for relationships, such as organization, generalization, reliance, place, and awareness the following:

\[
\text{A ELM } = \{\text{attr, oper}\}
\]

\[
\text{REL } = \{\text{assoc, gener, depd, aggr, realz}\}.
\]
Class represents a set of classes. PN is a set of prime numbers. For example

\[ \text{PN} = \{2, 3, 5, 7, 11, 13\}. \]

Definition 3.1 (Class-to-Class Relation): The relation between two classes is a function

\[ r : \text{Class} \times \text{Class} \rightarrow 2\text{REL}. \]

Example 3.1 (Class-to-Class Relation):

\[ r(A,B) = \{\text{assoc, gener, depd}\} \]

\[ r(A,B) = \{\text{aggr}\} \]

\[ r(A,B) = \{\text{realz}\} \]

represent that classes A and B have the association, generalization, and dependence relationships, aggregation relationship, or realization relationship, respectively.

Definition 3.2 (Encoding Function): The elements of sets ELM and REL are assigned with a unique prime number by a function

\[ \rho : \text{ELM} \cup \text{REL} \rightarrow \text{PN}. \]

Example 3.2 (Encoding Function): We use the following encodings:

\[ \rho(\text{attr}) = 2 \]

\[ \rho(\text{oper}) = 3 \]

\[ \rho(\text{assoc}) = 5 \]

\[ \rho(\text{gener}) = 7 \]

\[ \rho(\text{depd}) = 11 \]

\[ \rho(\text{aggr}) = 13. \]

Definition 3.3 (Cell Value Function): All relationships between two classes are mapped into an integer by a function

\[ \gamma : \text{Class} \times \text{Class} \rightarrow \mathbb{N} \]

such that \( \forall A, B \in \text{Class} \)

\[ \gamma(A,B) = i \]

\[ \rho(Ri) \forall Ri \in r(A,B) \text{ if } r(A,B) = i \]

\[ \gamma(A,B) = 1, \text{ if } r(A,B) = i. \]

Definition 3.4 (System Matrix): The relationships between the classes of a system are defined as a square matrix

\[ A_m = (a_{ij})_m \]
where $a_{ij} = \gamma(C_i, C_j)$, $C_i, C_j \in \text{Class}$, $1 \leq i, j \leq m$, and $|\text{Class}| = m$.

**Definition 3.5 (Weight):** The weight of each class is defined as a function

$\omega : \text{Class} \rightarrow \mathbb{N}$

such that $\forall A \in \text{Class}$

$\omega(A) = \rho_m(\text{attr}) \times \rho_n(\text{oper}) \times i$

$\gamma(A, C_i)$

where $1 \leq i \leq l$, $C_i \in \text{Class}$, $\rho_m(\text{attr})$ is the number of attributes that class $A$ contains, $n$ is the number of operations that class $A$ contains, and $l = |\text{Class}|$.

**Definition 3.6 (System Weight Vector):** The weights of all classes in a system are defined by the following vector:

Vector $B_m = (b_i)_m$

where $b_i = \omega(C_i)$, $C_i \in \text{Class}$, $1 \leq i \leq m$, and $|\text{Class}| = m$.

Consider a set of design patterns that need to be discovered from a software system $\text{PATTERN} = \{\text{adapter, bridge, strategy, composite, \ldots}\}$.

**Definition 3.7 (Pattern Class):** All classes that participate in a design pattern are defined as its pattern classes, and $\forall p \in \text{PATTERN}, \text{Class}(p)$ represents the set of classes participating pattern $p$.

**Definition 3.8 (Pattern Matrix):** The relationships between the classes of a design pattern are defined as a square matrix

$DM(p) = A_m = (a_{ij})_m, p \in \text{PATTERN}$

where $a_{ij} = \gamma(C_i, C_j)$, $C_i, C_j \in \text{Class}(p)$, $1 \leq i, j \leq m$, and $|\text{Class}(p)| = m$.

**Definition 3.9 (Pattern Weight Vector):** The weights of all classes in a design pattern are defined by the following vector:

Vector $DW(p) = B_m = (b_i)_m, p \in \text{PATTERN}$

where $b_i = \omega(C_i)$, $C_i \in \text{Class}(p)$, $1 \leq i \leq m$, and $|\text{Class}(p)| = m$.

**Definition 3.10 (Matrix Match):** Consider a system matrix $A_m = (a_{ij})_m$, $|\text{Class}| = m$, and a design pattern $p \in \text{PATTERN}$, $|\text{Class}(p)| = n$ with its matrix $DM(p) = (d_{ij})_n$. If there exists a submatrix of $A_m$ sub $A_n = (s_{ij})_n = A[k_1, k_2, \ldots, kn; k_1, k_2, \ldots, kn]$, $(1 \leq k_1, k_2, \ldots, kn \leq m)$ such that
\[ \text{sij mod dij = 0, } 1 \leq i, j \leq n \]

then the pattern matrix matches the system matrix.

**Definition 3.11 (Weight Match):** Consider a system weight vector \( B_m = (b_i)_m, |\text{Class}| = m \), and a design pattern \( p \in \text{PATTERN}, |\text{Class}(p)| = n \) with its weight vector \( \text{DW}(p) = (d_i)_n \).

If there exists a subvector of \( B_m \)

\[ \text{sub } B_n = (s_i)_n = B[k_1, k_2, \ldots, k_n], \]

\( (1 \leq k_1, k_2, \ldots, k_n \leq m) \)

such that

\[ \text{smod } d_i = 0, 1 \leq i \leq n \]

Then the pattern weight vector matches the system weight vector.

**Description 2.12 (Pattern Structure Match):** When both matrix and fat of the style pattern match these of the program, it is described as structure match. This description could be produced straight in the preceding two meanings. Informally speaking, the preceding definitions use dumbbells and matrices to signify the structural details of designs and methods and explain the coordinating of a design framework with a program framework. More especially, the program matrix (Definition 2.4) or design matrix (Definition 2.8) explains the associations, including generalization and organization, between the groups in a system or a design, correspondingly. Design matrix and dumbbells function while the requirements of architectural evaluation. If you find a matrix match (Definition 2.10) between a program matrix and a design matrix, it reveals that the system contains some groups having the same associations as these in the design. If the weights of these courses in the program additionally complement those of the classes in the design, which can be called a fat match (Definition 2.11), it reveals that these classes have the essential amounts of characteristics and procedures by the related design. Consequently, these groups, whose matrix and weights match those of the design, can be viewed like a framework match with the design (Definition 2.12) and, consequently, a candidate case of the design.
Consider a good example of an incomplete layout from the Java.awt package about the graphical interface style as proven in Fig. 2.1. This course plans of the incomplete style could be reverse-engineered from the source code of Java.awt by present reverse engineering programs for example IBM Rational Rose. Existing plug is used by us-ins of the resources, such as UML2Tools, to create the XMI representations of the design blueprints, because most of these resources continue design blueprints in some amazing structure. In this method, we are able to parse the XMI documents of the design blueprints and create the dumbbells and matrix of the machine design depending on this study strategy officially defined previously. The established Group comprises all groups of the look shown in Fig. 2.1. Desk 2.1 exhibits the matrix of the style created according to Description 2.4. This matrix encodes the connections between each set of groups with several. The object-oriented relationships between
groups are secured by a perfect number depending on Description 2.2 and Instance 2.2. The worth of the related cell of the matrix is going to be the product of most the corresponding perfect numbers coding the individual associations, when two groups have several associations. The associations between each set of the groups could be readily retrieved from their related cell value of the matrix, because every mobile worth is an intrinsic multiple of prime figures which signifies a distinctive blend of perfect figures. The value of every mobile in the program matrix may be used to decode the associations between the related set of groups.

For instance, there's a relationship from the course to the Element course. Therefore, the worthiness of the mobile about the very first strip and fifth line of the matrix is seven, as proven in Table 2.1. For asymmetric relationships, we handle incoming relationships and confident relationships otherwise because both finishes of the relationships are distinct. Likewise, you will find the organization and generalization relationships from the Pot class for the Element group, which trigger the worth of the related mobile to be 35 = 5 seven. There's an organization connection in the Element course to the Pot class, producing the accompanying mobile worth to five. With this kind of coding technique, it's fairly simple to decode the associations between any set of courses within the matrix. When the value of the mobile is 35, which can be the product of 7 and 5, for instance, then it's simple to decode that the accompanying set of courses has both generalization associations and organization. As described in Description 2.5, the fat of the course signifies the architectural features of the course, including the amounts of its traits, approaches, and confident associations that are secured based on Description 2.2 and Instance 2.2. For instance, the fat of the Fabric course in Fig. 2.2 is 23 35 seven = 13608. Remember that because the courses of each style pattern normally only comprise of a little number of characteristics, methods, and associations an optimisation of this strategy would be to just think about a optimum of five for the amount of characteristics, techniques or associations of a course. It's not very effective to depend the amounts of all approaches, characteristics, and associations in a course. This strategy counts 5 because of its fat, even though methods are included 11 by the Canvas class. The weights of additional courses in Fig. 2.2 could be calculated likewise. Let's consider the Composite design whose class diagram is proven in Fig. 2.2. Main role classes are contained three by it, specifically, Component, Composite and Leaf where the Composite and Leaf classes inherit from the Component class.
The matrix of the Composite design, as proven in Table 2.2, could be computed according to Description 2.8. The worth of every cell encodes the associations between the related set of groups. Likewise, the dumbbells of the Part, Composite, and Leaf courses could be calculated centered on Description 2.9, which are 81, 2835 and 21, correspondingly.

Based on Definition 2.10 and Definition 2.11, the architectural evaluation (Definition 2.12) contains the coordinating from the matrix and dumbbells of the layout, e.g., the Composite design, for the matrix and weights of the program, e.g., the incomplete layout of Java.awt proven in Fig. 2.1. For instance, the subwoofer matrix, including the Element, Box, and Switch lessons, could be considered a complement of the matrix of the Amalgamated design according to Description 2.10. Their related weights also fit these of the Composite design. Therefore, these three courses type a candidate case of the Amalgamated design according to Description 2.12.
2.3 BEHAVIOURAL ANALYSIS

Since it focuses on the structural part of style designs False positive cases may be contained by the result of this structural analysis. Most style designs have behavioral features as well as the architectural types. These behavioral qualities might contain control flows, e.g., some techniques require to produce some additional techniques. Our behavioral analysis appears for special method invocations in the program source code. The analysis is performed after the architectural analysis by coordinating the behavior characteristics described within the design description java document. Rather than looking the whole supply signal, the analysis considers just the courses in the prospect design examples acquired in the structural analysis. Therefore the research room could be reduced. Within the remainder of the area, the behavioral features are described officially by making use of predicates. The study strategy expose the meaning and format of the predicates found in these specs. Where the function titles are in line with the research then the behavioral features of Composite designs, and the Adapter, Technique are officially introduced. As mentioned in [14] the qualities of additional designs could be displayed likewise. Along with the formal standards, the Adapter pattern is used by this approach for example to demonstrate how the behavior evaluation, which is an execution akin to the formal specs of behavioral features, is done. For simpleness the execution information on additional designs are overlooked.

Description 2.13 (Syntax and Meaning of Predicates):

a) methodList(A) where A Course: the group of techniques in Course A.

b) name(methodi): the title of method methodi.

c) parameter(methodi): the listing of parameter kinds of method methodi.

d) returnType(methodi): the return type of method methodi.

e) call(methodi, methodj): methodi invokes methodj.

f) containSubstring (stringi, stringj): stringi contains stringj as its substring.

g) caseIgnore(stringi): Change all of the figures in stringi into lower case.

i) Adapter Pattern: In the Adapter pattern, there will probably be a standard approach, called Request, defined in the Goal and Adapter courses. The Request technique in the Adapter class shall phone a method, called Particular Request, described in the Adaptee class. These behavioral features of the Adapter pattern are formally described the following. Definition 2.14 (Adapter Behavior):
Generic Algorithm with Mathematical Evaluation for Quality Assurance of Real Time Applications

The Design Patterns

methodi methodj methodk
(1) methodi methodList(Target)
(2) methodj methodList(Adapter)
(3) methodk methodList(Adaptee)
(4) name(methodi) = name(methodj)
(5) parameter(methodi) = parameter(methodj)
(6) returnType(methodi) = returnType(methodj)
(7) call(methodj,methodk)

Behavioural evaluation is conducted based on the official standards of design behavioural features. According to predicates (1), (2), and (3), it gets a listing of techniques from the course which performs the part of Goal, a list of methods from the class which plays the role of Adapter, and a list of methods from the class which plays the role of Adaptee. It keeps the approaches which come in both listings and then filters the first two listings. This is based on examining the title predicate (4), the parameter predicate (5), and the get back type predicate (6) of the approaches. Lastly, according to predicate (7), this behavioral evaluation checks whether there's a technique in the maintained list that calls a method in the 3rd list. The behavioural analysis is passed by this pattern candidate, when there is such a procedure. Otherwise, it's a fake positive case of the Adapter design.

ii) Bridge Pattern: There will be one or more typical approach, called OperationImp, between the Implementer and ConcreteImplementor in the Link design. Lack of such approach suggests that there's no variance within the execution. This doesn't comply with the meaning of the Link design. The behavior of the design additionally demands that the Abstraction category shall produce one or more of the OperationImp approaches.

iii) Strategy Pattern: There will be one or more typical approach, called AlgorithmInterface, which is described within the Technique and ConcreteStrategy groups. The behavior of Technique additionally demands that the Framework group will produce one or more of the AlgorithmInterface approaches.

Description 2.16 (Strategy Behavior):
methodi methodj methodk
(methodi methodList(Circumstance)
methodj methodList(Strategy)
methodk methodList(ConcreteStrategy)
name(methodj) = name(methodk)
parameter(methodj) = parameter(methodk)
returnType(methodj) = returnType(methodk)
call(methodi,methodj)).

iv) Composite Pattern: The Amalgamated design demands that one or more typical approach, called Operation, shall exist among Leaf lessons, Composite, and Element. The procedure within the Leaf and Amalgamated classes will bypass that of the parent. The Operation method shall be invoked by the Operation method in the Composite class within the Element class. In the Amalgamated course, there will be considered a method enjoying the part of the Include or Remove method which has a parameter with as its kind the Element.

Definition 2.17 (Composite Behaviour):

\[ \exists \text{methodi} \exists \text{methodj} \exists \text{methodk} \]
\[ (\text{methodi} \in \text{methodList(Component)} \land \text{methodj} \in \text{methodList(Composite)} \land \text{methodk} \in \text{methodList(Leaf)} \land \text{name(methodi)} = \text{name(methodj)} \land \text{parameter(methodi)} = \text{parameter(methodj)} \land \text{returnType(methodi)} = \text{returnType(methodj)} \land \text{name(methodj)} = \text{name(methodk)} \land \text{parameter(methodj)} = \text{parameter(methodk)} \land \text{returnType(methodj)} = \text{returnType(methodk)} \land \text{call(methodj,methodi)} \land \\
\exists \text{methodl(methodl} \in \text{methodList(Composite)} \land \text{parameter(methodl)} = \text{Component}). \]
2.4 SEMANTIC ANALYSIS

Some style designs resemble one another in their architectural and behavioral elements, for instance, the Link and Technique designs. Behavioral and architectural studies aren't enough to recognize these sorts of designs because they vary merely in their motives and intents. Such semantic distinction is normally difficult to test. Nonetheless, we discovered that many designers follow some labeling conventions when they provide the titles to classes, procedures, and characteristics. Such naming conventions occasionally might depart some track of the first style intents. For instance these architectural and behavioral studies of the Java.awt package return the same candidate case established for the Link and Technique designs. One of them, some courses are named BufferStrategy, FlipBufferStrategy, SimpleBufferStrategy and BltBuffer-Strategy. This is often a great indicator these are now prospects of the Technique, instead than Bridge design. The design qualities in an XML document is given which comprise the semantic characteristics of a design. This study strategy formally the following determine such semantic evaluation of the naming convention for the Technique design. The Link semantics could be described likewise.

Description 2.18 (Strategy Semantics):
containSubstring(caseIgnore(name(Context)),
caseIgnore("strategy"))
containSubstring(caseIgnore(name(Strategy)),
caseIgnore("strategy")) containSubstring(caseIgnore(name(ConcreteStrategy)),
caseIgnore("strategy")) (methodi)
methodi (methodList(Context)
methodList(Strategy)methodList(ConcreteStrategy))
containSubstring(caseIgnore(name(methodi)), caseIgnore("strategy"))).