UNIT - V

Class design :-

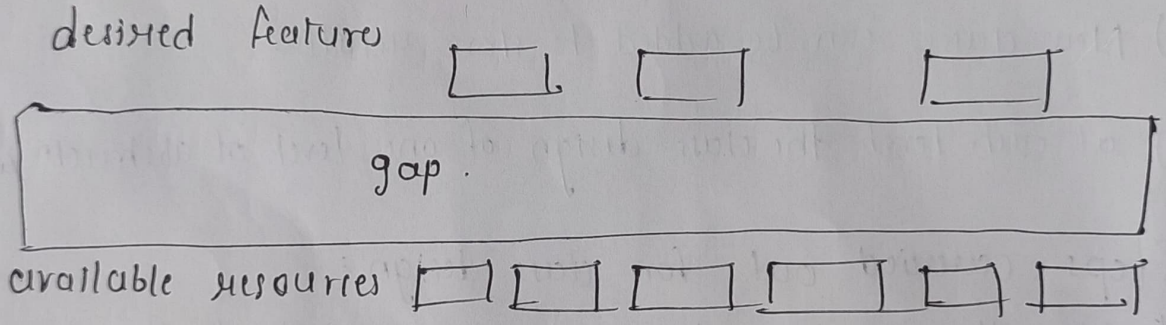
- During the analysis phase the class design is created.
- Purpose of class design is to complete definitions of classes & associations & choose algorithm for operations.
- class design created in analysis model is directly mapped into design.
- during design main task of class design is choosing appropriate algorithms for operations of classes & breaking the complex operations into simple.
- New classes can be added to store results.
- at each level the class design of one level of abstractn is created.

Steps carried out for class design :

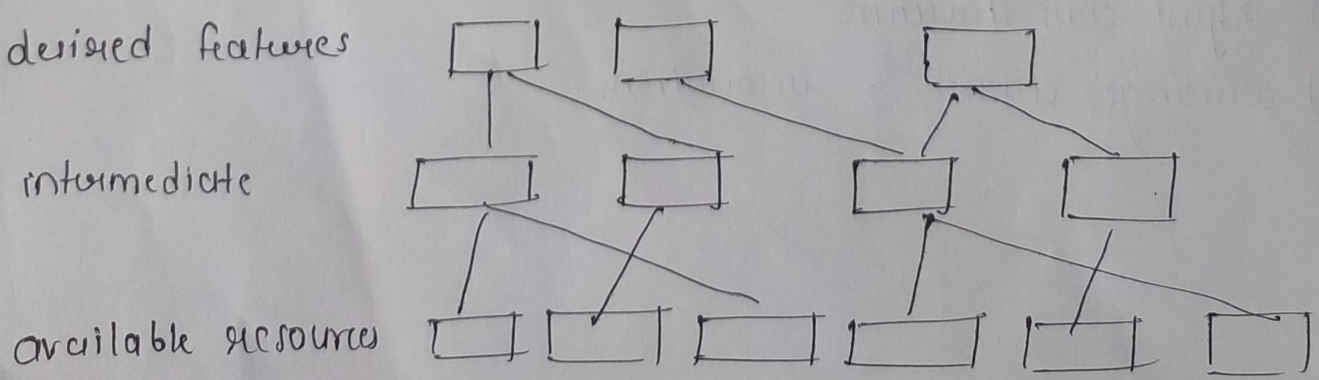
- Bridge the gap blw high level requirements to low level services.
- analyze use cases
- for each operation design an algorithm
- Recurse downwards to design operations
- Refactor the model
- optimize path
- Refy behaviour
- adjust class structure
- arrange classes & associations.

☆ Bridging the Gap :-

- Every system has its own set of features.
- System offer service when features are utilized by resources available.
- But there exists a gap blw desired feature & available resource. that why designer has to build a bridge across gap.
- Features can be obtained from usecases, system operations & services etc.
- Resources can be OS, class libraries.



- Making use of class libraries or databases so that the features can be made available as a service called bridging the gap.
- sometime bridging gap is difficult. so developer has to invent some intermediate elements so that features can be used by available resources to provide services.
- sometimes intermediate elements are hard to find.

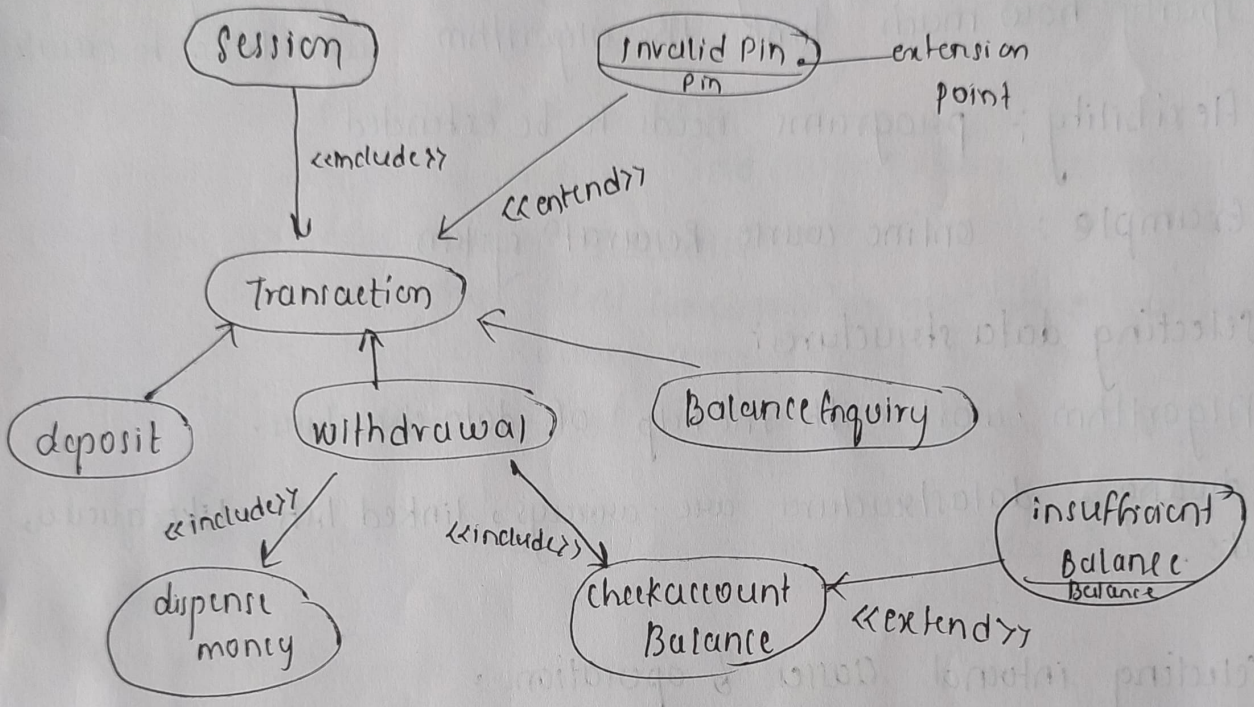


Realizing use cases :-

use cases defines behaviour of system.

- during system design, new operation & new object can be identified provides behaviour.

- Example :-



withdrawal of money from ATM.

★ Designing Algorithms :-

Steps to design algorithm :-

-) select algorithm that reduces overall cost of operations.
-) select datastructure for implementing algorithm.
-) defines new internal class & operations if required.
-) assign operations to class

1) Selecting algorithms :-

factors considered while choosing alg :-

·) Ease of implementation & understandability :- alg. must be simple.

·) computational complexity :- Denotes efficiency of algorithm. It specify how much time the algorithm will take to execute.

·) Flexibility :- programs needs to be extended.

·) Example :- online course Reservation system.

2) Selecting data structures :-

·) Algorithm work with the help of data structures.

·) ~~during~~ datastructures are arrays, linked lists like queues, stacks.

3) Selecting internal classes & operations :-

·) high level operations need to be decomposed into low level operations.

·) due to expansion of alg new classes can be added & hence new operations get introduced.

·) for ex :- new class Receipt can be introduced when fees payment is accepted.

4) Assigning operations to classes :-

·) during design, new internal classes are introduced. to find name of class that contain certain operation, we need to identify object & collect info about operations perform by these objects

Recursing Downwards :-

operations at higher layers invoke the operations at lower layers.

- downward recursion can be done either by functionality & mechanism layers.

•) functionality layers :-

→ it means the high level functionality is broken into small functions

→ this decomposition must be logical.

→ combine similar operations and attach them to classes.

→ dangers in functionality recursion :-

·) if higher level functionalities are broken arbitrarily then the lower level operations can't related to classes.

·) this depends heavily upon top level functionality.

→ To avoid this danger, attach operations to classes & make them more useful.

•) Mechanism layers :-

→ it means building system to support various much needed mechanisms.

→ various mechanisms that system needs are - storing information, coordinating objects, transmitting information.

→ It proride the support to perform high level responsibilities.

→ In large systems functionality layers are mixed with mechanism layers.

→ If system is designed using functionality layer then it becomes very sensitive while if system is designed using mechanism layer the it becomes less useful. Hence the use of these layers is always preferred.

Refactoring :-

·) As design evolves the operations and classes get identified & their use is getting understood by the developer.
Hence it is essential to revisit the design, rework on classes and operations & make them conceptually clear.

·) Refactoring means changes in internal structure of sw to improve design.

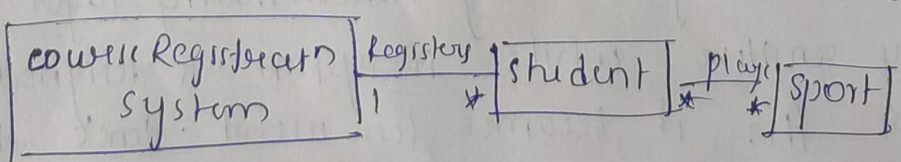
·) Although it is time consuming.

Design optimizatn :-

·) It is always good practice to create design & then optimize it. because simultaneously, creating & design & optimize it is difficult.

·) tasks carried out during optimizatn :-
- ·) adding efficient acces paths
- ·) Rearranging execution order
- ·) saving derived value.

·) adding Redundant associations :-
- they are utilized in such a way that the efficient access to other classes can be possible.
- Redundant associations do not add any useful info. to analysis model. hence it is not present in analysis model.
- Example :- consider online course Reservation system



) If there are 1000 students, out of which only 10 students have skills for sports then for finding the sports persons a simple loop execute for 1000 times. This lead inefficient execution. to eliminate this problem, we can create hashed set for play instead of unordered linked list.

•) factors considered while adding the redundant associations :-

→ frequency of access : how many times the operatn is accessed.

→ selectivity : It is based on hits. hits is a number that denotes finding out of target object.

→ fanout : average count on each many associations that are calculated in traversal path.

•) Rearranging execution order :-

•) It includes elimination of dead paths, many times it can be inherited from original specification for optimization.

•) Saving Derived values :-

•) New classes can be added for storing derived attributes & avoid the recomputation.

•) this cache needs to be updated.

•) ways to handle updates of cache :-

→ explicit update : explicit code is added to operatn of source attributes

→ periodic recomputatn : derived attributes can be recomputed periodically because app changes the values.

→ active value : value which is auto kept consistent with its source value.

Reification of Behaviour :-

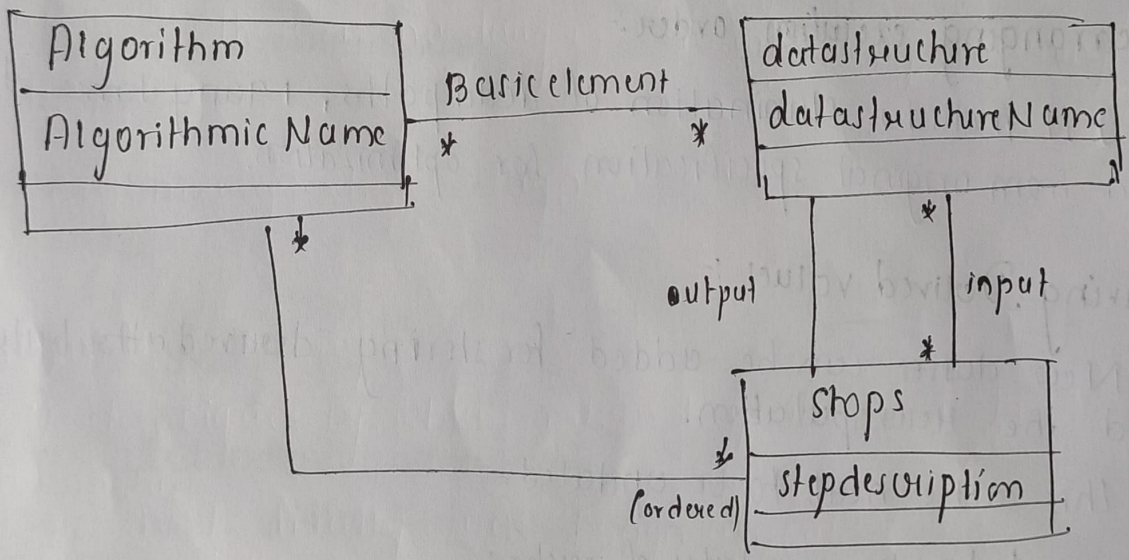
•) Manipulation of behaviour in code is not possible during runtime (execution). But sometimes it should be modified. so for that we need to reify the behaviour.

•) Reification is mechanism in which some entity which is not object is converted to an object.

•) If behaviour is reified then we can store it, pass it to another operation or modify it.

•) It increases the complexity but brings a flexibility in system.

•) behavioral patterns that are used to reify the behaviour. are state, commands & strategies.



•) State represent runtime interpreter.

•) command represent encoding sequence of requests.

•) strategy represents operations.

Adjustment of Inheritance :-

Inheritance can be adjusted using following steps :-

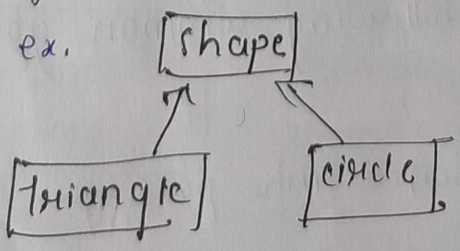
1) Rearranging classes & operations :-

-) Some times several classes define same operations, by adjusting the operations, single inheritance can be adjusted.
-) all operations must have same semantic & signature.
-) adjustments can be done to increase inheritance are :-
  - → operations having special cases
  - → operations with optional argument
  - → inconsistent names.
  - → irrelevant operations
-) Example :- online course registration system → student pays fees class payment can be supported by creditcard payment & cash payment.

2) Abstracting out common behaviour :-

-) whenever, there is common behaviour, then a common superclass can be created and specialized features can be added into subclass. This transformation called abstracting out the common behaviour.
- It is always good to have abstract super class & specialized subclasses

ex.



- super classes are useful for sharing & reuse.
- extensibility of s/w product can be improved.
- It is used in configuratn management & maintenance of s/w product because it generated customized versions of s/w.
- Example:- If we keep payment class as abstract then it is specialized by adding subclasses like chequepayment, COD payment & creditcardpayment.

3) using Delegation :-

-) Delegation is a mechanism in which only useful operations from one object are cached & send them to another object.
-) If we use delegation it only delegate the meaningful operations & there is no danger of inheriting meaningless operations.

organizing class design :-

Steps :-

1) Information hiding :-

-) technique by which internal specificatn is hidden from outside world.
-) don't directly access foreign attributes.
-) limit the scope.
-) hide external objects.

2) coherence of entities :-

-) package is coherent if it is organized on consistent plan and all part of design are following common goal.

3) fine tuning packages :-

-) class model is partitioned into packages.

package should have interfaces.
- Interface b/w packages must be associatn.
- package should have some specific purpose.

Implementation Modeling :-

-) It is final stage of development.
-) Suitable prog. lang is used to create working model.
-) It must be straightforward & almost mechanical.

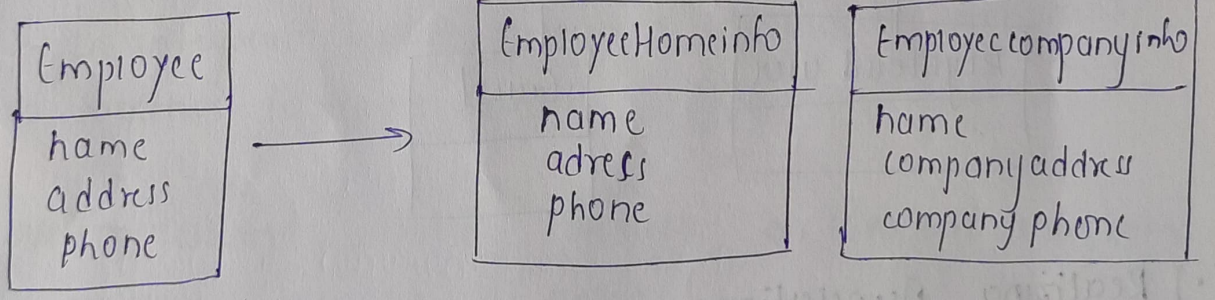
Steps :-

(•) fine tuning classes :

-) it makes code simple to understand & increases the performance of execution.
-) techniques used to fine tune the classes :-

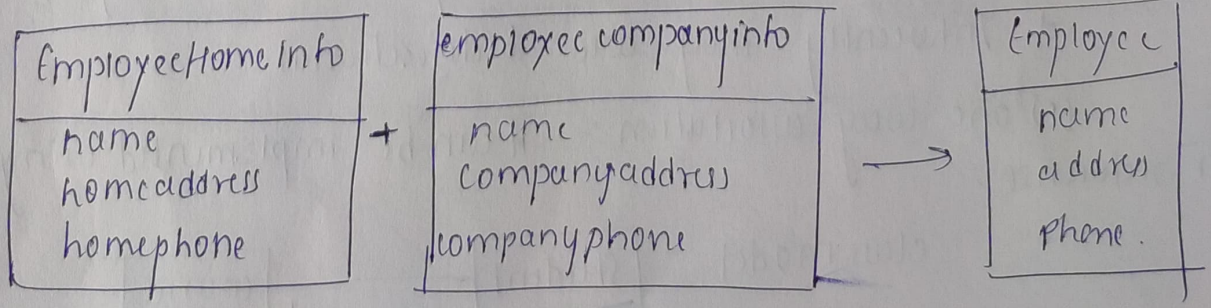
(1) partition a class :- when class is partitioned the info of single class is split into two classes.

Ex :-

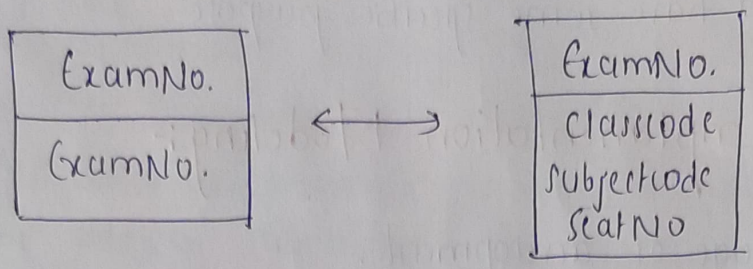


(2) Merge class :- converse of partitioning is merging.

Ex :-



(3) partikon or merge attributes :- when class gets fined tuned sometimes we can adjust attributes by partikoning or merging.

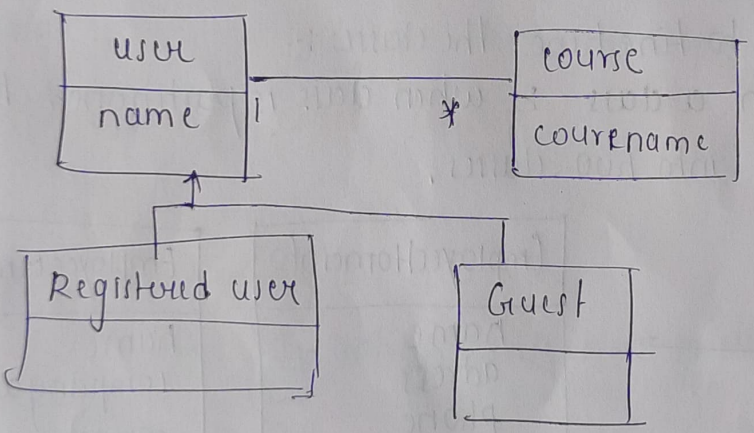


(4) promote an attribute :-

(·) entity can be represented as attribute or class.

(·) Fine tuning Generalisations :-

→ it means either adding generalization or removing it before coding.



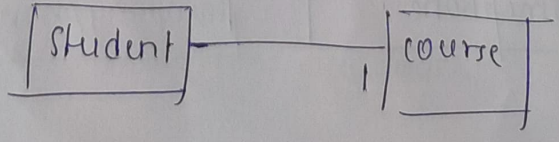
(·) Realizing Associations :-

→ Association provide access path blw two objects.

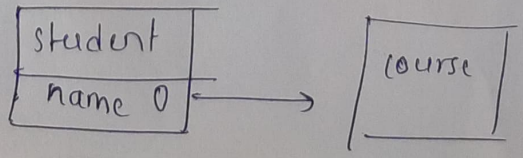
→ Inherently they are bidirectional.

(1) one way association :- it can be implemented using pointer or reference.

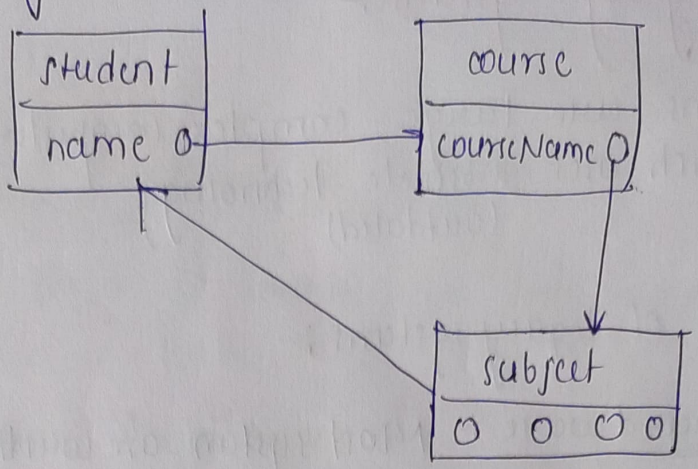
class model



implementatn



(2) Two way association: Implementation using two way *.



(•) Testing :-

→ It is quality assurance mechanism used for catching errors.

→ Number of bugs identified give measure quality of slw.

→ It should be carried out at every stage.

* unit testing :- •) This techniques are applied to detect the errors from each slw individually.

•) It focus is to uncover errors in design & implementatn

•) In integratn testing, group of dependent components are tested together to ensure quality.

* system testing :- •) Testing conducted on complete integrated system to evaluate working of system

•) It is tested in context of functional & system requirement

•) Functional testing focuses on actual usage of product.

•) Non functionaltesting focuses on customer expectations.

Legacy Systems :-

- Legacy systems are large complex computer based systems which uses obsolete technology (outdated)

- components of legacy systems :-

  ✱ system hardware :- Most system are written for mainframe computers.

  ✱ supporting slw :- This are older os, compilers & utility slw.

  ✱ App slw :- It includes separate programs that include business logic.

  ✱ App data :- app data is accumulated & then used. Stored in files.

  ✱ Business process :- To meet business objective process are executed.

  ✱ Business policies & rules :- include set of rules.

Legacy system layered Technology :-

- Interfaces are maintained for each layer of system so that if one layer change then it should not affect the other layer.
- But in reality, it changes the other layer because :-

  1) If one layer change, new facilities are added. To take advantage of new facilities the other layer demand for change.

  2) change in slw may cause degradation in performance

Techniques that are used to deal with legacy systems :-

Reverse Engineering :-

•) It is the process of design recovery.

•) In this, the data & architectural info is extracted from source code.

→ Input to Reverse Engineering :-

(1) programming code → It helps to understand flow of control & data structure.

(2) data : data structure is discovered.

(3) database structure :-

(4) forms & reports.

(5) documentation.

(6) app understanding

(7) test cases.

→ output from Reverse Engineering :-

(1) Models : represent sw scope & intent : used for understanding original sw.

(2) Mappings :- Mapping of prog. code to state or ~~integration~~ interactn model.

(3) logs : document that ~~models~~ records the observations & pending questions.

*) During Reverse engineering process, class model, statemodel, interaction model are build.

Building class Models :-

→ Reverse engineering begins from class model.

→ phases in building class model are :-

1) implementation recovery :-

-) from app, create an initial class model.
-) if program written in oop lang then class & generalizations are directly used in initial class model.
-) if program is not in oop lang., then data struct & operatn are studied.

2) design recovery :-

-) the next step is to recover associatn.
-) multiplicity in one direction is identified using single pointer attribute in implementatn.

3) Analysis recovery :-

-) firstly proper interpretation of model is done, then it is refined & made more abstract.
-) all redundant informatn is eliminated.
-) aggregation & composition is identified.

Building interaction Model :-

-) for interaction model, it is necessary to understand behaviour of system.
-) consider class model & add methods using slicing technique. A slice is subset of program that represent specific behaviour.

The slice code is converted from procedural to oop representation

·) activity diagram is build to represent flow of data.

·) then sequence diagram can be build for simplification purpose

Building state Model :-

·) for UI the state model is used.

·) for building state model, sequence diagrams are taken as input.

·) all possible states are identified.

Useful tips for Reverse engineering :-

1) use flexible process.
2) Expect multiple interpretations.
3) Expect odd constructs.
4) watch for consistent style.

Wrapping :-

·) A wrapper is collection of interfaces that control access to system

·) It consists of set of boundary classes that provide interfaces

·) legacy code is complex & messy but boundary classes of wrappers hide the details from outside.

Maintenance :-

1) initial development :- developer create slw.
2) Evolution :- slw undergoes changes
3) servicing
4) phase out.
5) close down