Page :
Date : / /

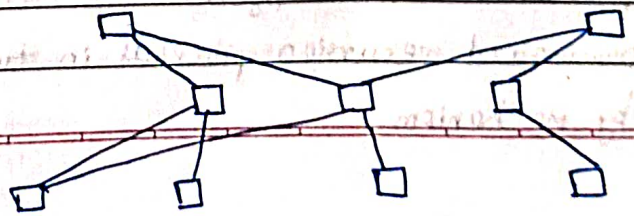
OOMD Unit 5

Q. Class design -

→ - Class design is a process of adding detail and making fine decisions.
- We choose among different ways to realize analysis classes with an eye towards minimizing execution time, memory and other cost measures.
- This decomposition is iterative process that is repeated at successively lower levels of abstraction.
- Class design involves following steps -
1. Bridging the gap.
2. Realizing use cases.
3. Designing algorithms.
4. Recursing downward.
5. Refactoring.
6. Design optimization.
7. Reification of Behaviour
8. Adjustment of Inheritance
9. Organizing Class Design.

Q. Bridging the gap -

→ - In this step, we bridge the gap from high-level requirements to low level services.
- Sometimes, it is not possible to readily build the system due to too much gap betn resources and features.
- So, we need to invent intermediate elements to bridge this gap.
- These intermediate elements may be operations, classes or other UML elements.



Page :
Date : / /

Q. Realizing use cases-

→ - During class design, we elaborate complex operations, most of which come from use cases. Use cases define required behaviour, but they do not define it's realization.

- Use cases provide system-level behaviour. During design, we need to invent new operations and new objects that provide this behaviour.

- We list responsibilities of use case or operation. A responsibility is something that an object knows or something it must do. eg. in online theater ticket system, making a reservation has responsibility of finding unoccupied seats, obtaining payment from customer, delivery of tickets, etc.

- Each operation will have various responsibilities. Some of these may be shared by other operations and others may be reused in ~~fut~~ future.

- Define operation for each responsibility cluster and assign new lower-level operations to classes.

Q. Designing Algorithms-

→ - In this step, we formulate an algorithm for each operation.

- The analysis specification tells what operation does for it's clients, but algorithm shows how it is done.

- Algorithms are designed using following steps-

1. Choose algorithms that minimize cost of implementing operations.
2. Select appropriate data structures.
3. Define new internal classes and operations as necessary.
4. Assign operations to appropriate classes.

Q. Recursing Downwards-

→ - We organize operations as layers - operations in higher layers invoke operations in lower layers.

- Downward recursion proceeds in two main ways - by functionality, by mechanism.

Page :
Date : / /

1. Functionality Layers-
- Functionality recursion means you take required high-level functionality and break it into lesser operations. It is a natural way to proceed.
- But it can cause trouble if we perform decomposition arbitarily. To avoid it, we combine similar operations & attach operations to classes.

2. Mechanism layers-
- It means we build system out of layers of needed support mechanism. we need various mechanisms to store information, co-ordinate objects, perform computations, etc.
- They are not directly part of user's needs for space, but they are needed to enable the chosen architecture.
- Any large system mixes functionality layers and mechanism layers.

Q. Refactoring-

→ - The initial design of set of operations will contain inconsistencies, redundancies and inefficiencies. It is difficult to get a large design correct in one pass.
- So, we define refactoring as changes to internal structure of software to improve it's design without altering it's external functionality.
- We step back, look at different classes and operations and re-organize them to support further development.
- Refactoring keeps a design viable for continued development and it is an essential part of good engineering process.

Q. Design Optimization-

→ - It is difficult to optimize a design at same time as we create it. So, good way to design system is to first get logic correct and then optimize it.

- Once we have logic in place, we can run application, measure performance and then fine tune it. Often, a small part of is responsible for most of the time and space costs.
- So, it is better to focus optimization on such critical areas to spread effort evenly.
- Design optimization includes following paths-
1. Provide efficient access paths.
2. Rearrange computation for greater efficiency.
3. Save intermediate results to avoid recomputations.

Q. Reification of Behavior-

→ - Reification is promotion of something that is not an object i an object. Behavior written in code is rigid.
- we can execute behavior but cannot manipulate it in run time we reify behavior by encoding it into an object and decod it when it is in run.
- After reification, we can store, pass or modify behaviour at r

Q. Adjustment of inheritance-

→ - As class design progresses, we can adjust the definitions of classes and operations to increase inheritance by following ste
1. Rearrange classes & operations.
2. Abstract common behaviour out of groups of classes.
3. Use delegation to share behaviour when inheritance is semantically invalid.

Q. Organizing Class Design-

→ - Programs consist of discrete physical units that can be edited compiled, imported or other-wise manipulated.
- So, to improve organization of class design, we can perform belo steps-

- Hide internal information from outside view.
- Maintain coherence of entities.
- Fine-tune definition of packages.

Q. Overview of Implementation-

→ - Implementation is final development stage that addresses the specifics of programming languages.
- Implementation modelling involves following steps-
- Fine-tune classes
- Fine-tune generalizations
- Realize associations
- Prepare for testing.

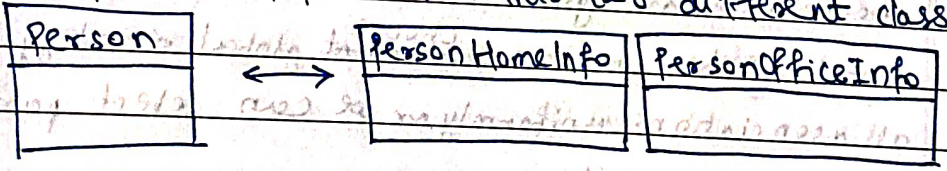
Q. Fine-tuning classes.

→ - Sometimes it is helpful to fine-tune a model by partitioning or merging classes. We fine tune classes in order to simplify development or to improve performance.
- It has following methods-

1. Partition a class-

- If we have home and office data for a person, it would better to split the information into two different classes.

eg.

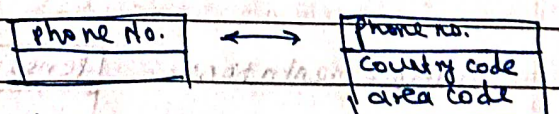


- But if we have modest amount of data, it may be easier to combine them.

2. Merge classes-

It is converse of partitioning a class. Give reverse example ↑

3. Partition/merge attributes

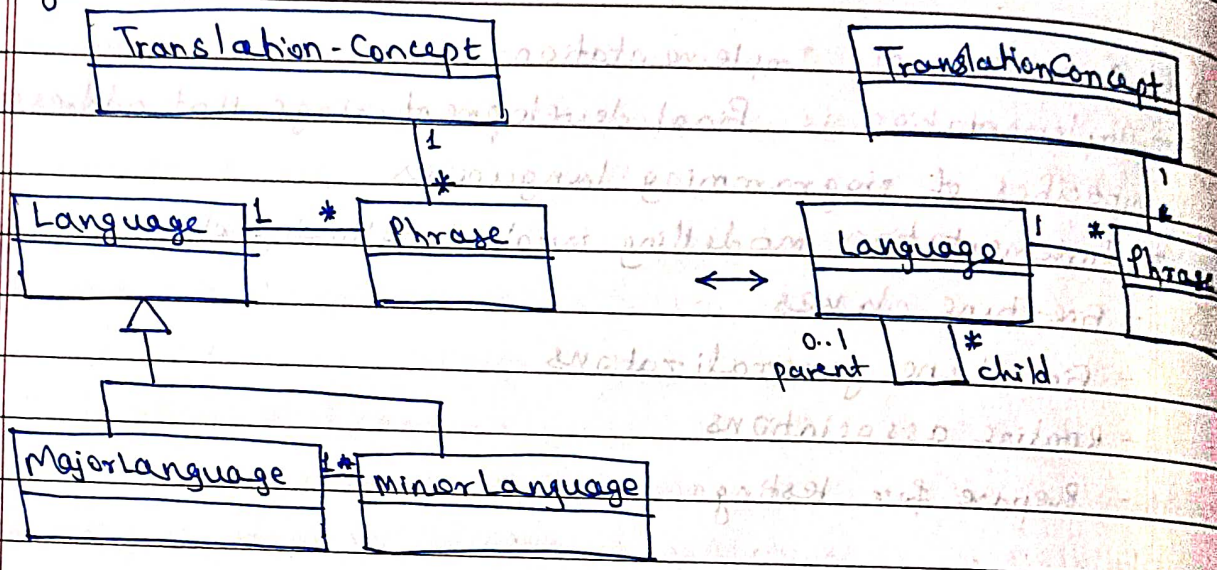


4. Promote attribute/demote a class.

Page :
Date : / /

Q. Fine-tuning Generalizations-

→ — As we reconsider classes, we can also reconsider generalizat
Sometimes, it can simplify implementation.
eg.



Above figure shows translation model which converts translation concept into phrase of desired language. We have two types of lang Major and Minor. So, for implementation simplicity, we ~~impr~~ rem the generalization and used the right model.

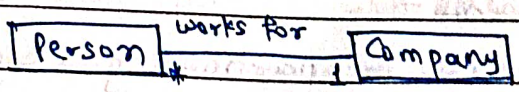
Q. Realizing associations-

→ — Associations are 'glue' of class model, providing access paths betw objects. Either we can ~~implement~~ choose global strategy for implementin all associations uniformly or we can select particular techniq for each association

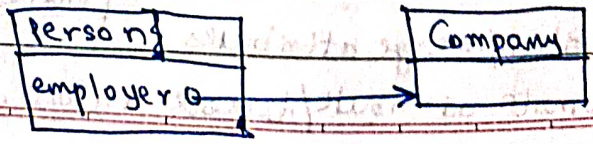
— Associations are inherently bidirectional but if association is traversed in only one direction, their implementation can be simplif

— If association is traversed in only one direction, we can implemen it as a pointer. eg.

Class model:



Implementation:



- When associations are traversed in both direction, we can implement one-way, implement two-way or implement with an association object.

Q. Testing-

→ - If we have carefully modelled our application, we reduce our errors in software and thus need less testing. Testing is a quality-assurance mechanism for catching residual errors.
- It provides an independent measure of quality of software. The number of bugs found for given testing is an indicator of software quality.
- Developers test from small pieces to ultimately the entire application. In unit testing, developers test their own code, their classes and methods.
- The next step is integration testing that is, how classes and methods fit together. It is done in successive waves, putting code together in increasing chunks of scope and behaviour.
- A separate team apart from developers carry out system testing. The scenarios of interaction model define system-level test cases.
- Once, alpha testing is complete, customers perform beta tests and then, if software looks good, it is ready for general release.

Q. Reverse Engineering-

→ - It is process of examining implementation artifacts and inferring the underlying logical intent.
- When building new applications, the purpose of reverse engineering is to salvage useful information. The reverse engineer must determine what to preserve and what to discard.
- Reverse engineer must overcome problems of program code like retrieving lost information and uncovering implicit behaviour. It is

Date : / /

inverse to forward engineering.
- While performing it, we must be resourceful and consid
all inputs like Programming code, Database structure, D
and reports, Documentation & Test Cases.
- It has several useful outputs like Models, Mappings
Logs.

Q. Building Class Model-
→ - Class model is built in 3 distinct phases-
1. Implementation Recovery-
- Learn about application & create initial class model.
- If program written in OO language, recover classes & gener
- Else study data structures & operations & manually defi
classes.

2. Design Recovery-
- Probe the application to recover associations.

3. Analysis Recovery-
- Finally, interpret model, refine it and make it more abst
- Remove remaining design artifacts & eliminate any error

Q. Building ~~State~~ Interaction Model-
→ - The way objects interact to carry out purposes of system is
often hard to understand from code. So, interaction model
a broad understanding.
- We can add methods to class model by using slicing. A slice is
subset of program that preserves specified behaviour of pro
- We can use activity diagram to represent extracted method
understand the sequence of processing and flow of data to
various objects.
- We can construct sequence diagrams from activity diagram
for simplification.

Page :
Date : / /

Q. Reverse Engineering Tips-

→ 1. Distinguish suppositions from facts.
2. Use a flexible process.
3. Expect multiple interpretations.
4. Don't be discouraged by approximate results.
5. Expect odd constructs.
6. Watch for consistent style.

Q. Wrapping-

→ - A wrapper is a collection of interfaces that control access to a system. When organizations limit changes to applications, they isolate the code and build a wrapper around it.
- Wrapping preserves the form of legacy software and accesses its functionality.
- Many existing applications have functionality that is confusing, complex and unpredictable. In this case, a wrapper provides clean interface for exposing the core functionality.
- If we are adding new functionality, it can usually be added as a separate package. We try to minimize interactions with the existing system.
- Wrapping is usually just a temporary solution, because wrapper is heavily constrained by organization of legacy software.

Q. Maintenance-

→ Much of the software moves through 5 stages of maintenance-
1. Initial development- Developers create software.
2. Evolution- Software undergoes major changes in functionality & architecture.
3. Servicing- Software changes are limited to minor fixes & simple functionality changes. Wrapping becomes appropriate strategy during this stage.
4. Phaseout- The vendor continues to receive revenue from

product but is now planning it's demise.
5. Closedown- The product is removed from market and customers are redirected to other software.

These five stages do not have a rigid wall between them, but there is continual decline in technical quality of software as it proceeds through it's lifetime.