# Final of CS322 (Close book)
# Dec. 20, Fall 2017

1. *Conceptual questions*

   - **(4 pts)** As a software engineer, what are your legal and ethics responsibilities when you work for medical and military applications?
   - **(4 pts)** What are the two levels of system design? Describe their corresponding main purposes briefly.
   - **(4 pts)** What is a CASE tool? Give one example and describe its importance in your software development efforts.
   - Compare the following concepts (state the respective definitions and the difference) **(2 pts/each)**:
     - data relation: association and cause-and-effect
     - test by experimentation vs test by observation
     - black-box testing vs white-box testing
     - software re-engineering vs. reverse software engineering.

2. *UML based design*

   Draw the E-R diagram **(5 pts)**, collaboration diagram **(5 pts)**, JSD tree **(5 pts)** and Petri-net **(5 pts)** for the following requirements in a video store:

   there are two types of videos: old and new, and two types of users: normal and special. A normal user can check out an old video for 2 weeks; while a special user can borrow one for 3 weeks. A late return will incur a fine measured by the number of weeks. A new video can be checked out by any user for three days. The fine for the later return is charged by number of days. Any user who returned any type of videos late more than twice will be levied an extra amount of money. Users who failed to return the video(s) for more than 6 months are reported to the collection company and put in the black list. Any user in the black list who wants to check out videos will be handled by the security officer.

3. *coupling and cohesion*

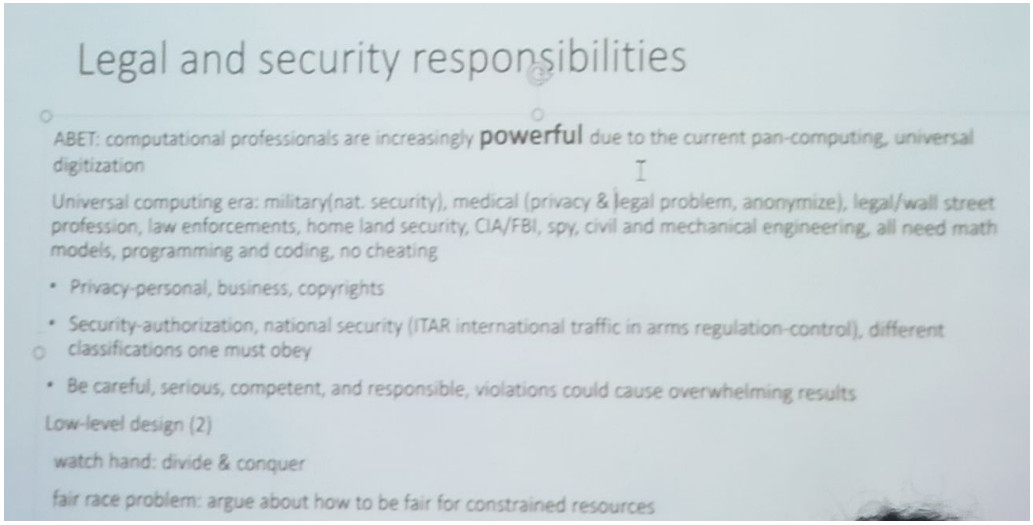
   (a) Label and justify the cohesion levels of the following modules **(2 pts/each)**:
   i. download a digital picture; play a digital video.
   ii. parse the search criteria; search the DB for matches.
   iii. locate the record to be updated from the DB; update the record; save the record back to the DB file.
   iv. design the system using OOD; implement the system using OOP.

   (b) Label and justify the coupling relation between every pair of the following modules (global and private variables are prefixed by g_ and p_, respectively) **(2 pts/each)**:
   Module a:
   p_a = (module) c(0, 1); if (p_a == 1) g_flag = 8;
   Module b:
   g_flag = 0; p_c = 0;
   Module c: parameter c1, c2;
   p_c=1; if (c1 < c2) p_c=8; if (c1 > 2) cout << c1 << endl;

4. **coding and analysis (13 pts)** Given an interger array A of size n that is already in ascending order, for a quary value v, find the five elements from A that are closest to v, that is, with 5 smallest possible $|A[i] - v|$. Pay attention to various special cases and *don't take anything for granted.* Indicate and prove the time complexity of your function.

5. **coding and testing (12 pts)** For two arrays A and B that are already in ascending order, write a program to mingle them into a new array ordered C *evenly*, the size of C is varied, depending on the values of A and B: the even-indexed (the index starts from 0) elements come from A while odd-indexed elements are from B. For instance, if A is {1, 4, 10,12}; B is {2,3,10, 11}; then the new array C is {1, 2,4,10,12}. If a black-box test approach will be used, how many test cases should you design?

- What are your legal and ethical responsibilities when you work for medical and military applications?
    - Be careful, be serious, competent and responsible, violations could cause overwhelming results.
    - Privacy: personal, business, copyrights
    - Security: authorization, national security (ITAR: international traffic in arms regulation-control), different classifications one must obey




    - High level: architecture design. Come up with modeling decisions
    - Lower Level: data structure+algorithm/logic

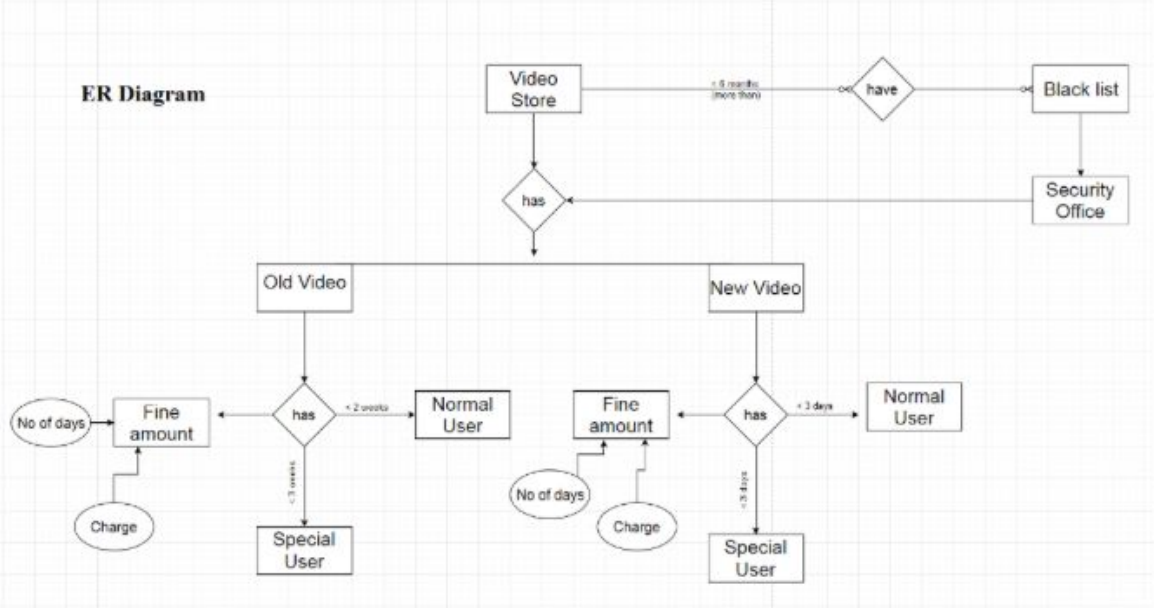
- What are the two levels of system design? Describe their corresponding main purposes.
    - High level: architecture design, the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer. Come up with moduling decisions
    - Lower Level: data structure+algorithm/logic - To represent the data flow, inputs and outputs of the system. Example: ER Diagrams (Entity Relationship Diagrams).
- What is a case tool?
    - CASE: computer aided software engineering. CASE tools are set of software application programs, which are used to automate Software Development Life Cycle (SDLF) activities.
    - **Examples:**
    - Diagram tools - These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.
    - Design Tools - These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using

refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design

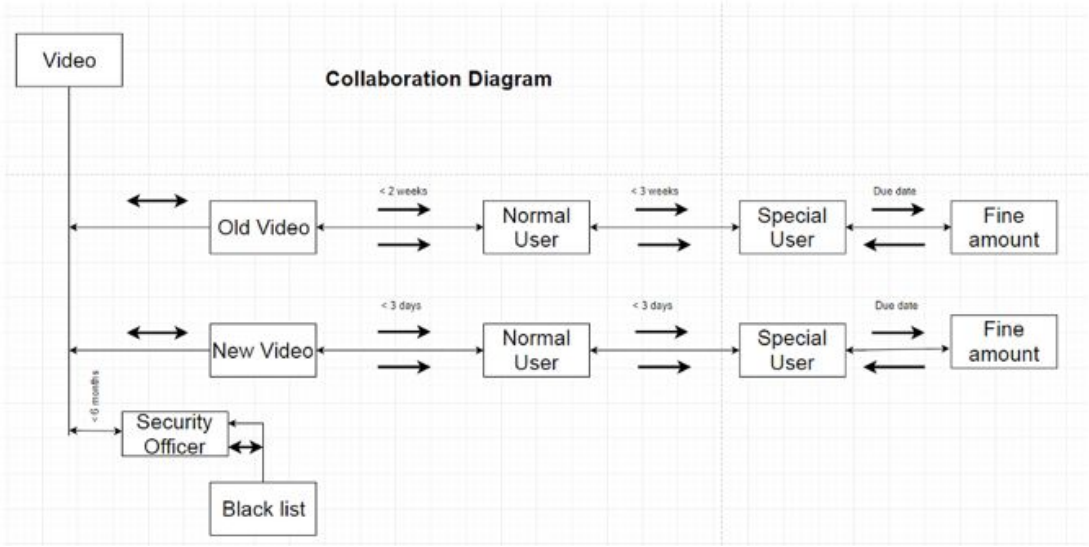
- Project Management Tools - These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply with project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Basecamp.
- Tools like Github for collaboration between group members. Text editors e.g Vim that help maximize the production of code

- Compare the following concepts:
- Data relation: association and cause-and-effect
  - Causation - When an article says that causation was found, this means that the researchers found that changes in one variable they measured *directly caused* changes in the other.
  - Association - When researchers find an association, what they are saying is that they found a relationship between two, or more variables.
  - Better explanation here
- Test by Experimentation vs Test by Observation
  - Experiment: in experiment investigators apply treatments to experimental units (people, animals, etc.) and then proceed to observe the effect of the treatments on the experimental units.
  - Observation - In an observational study investigators observe subjects and measure variables of interest without assigning treatments to the subjects. The treatment that each subject receives is determined beyond the control of the investigator.
- Black Box testing vs white box testing
  - Black Box testing - software testing method in which the internal structure/ design/ implementation of the item being tested is **NOT** known to the tester
  - White Box testing - internal structure/ design/ implementation of the item being tested **IS** known to the tester
- Software Re-engineering vs Reverse software Engineering
  - Software Re-engineering - tackling problem from scratch to improve and make corrections
  - Reverse Engineering - breaking down current item/idea to its individual parts to start a new production process

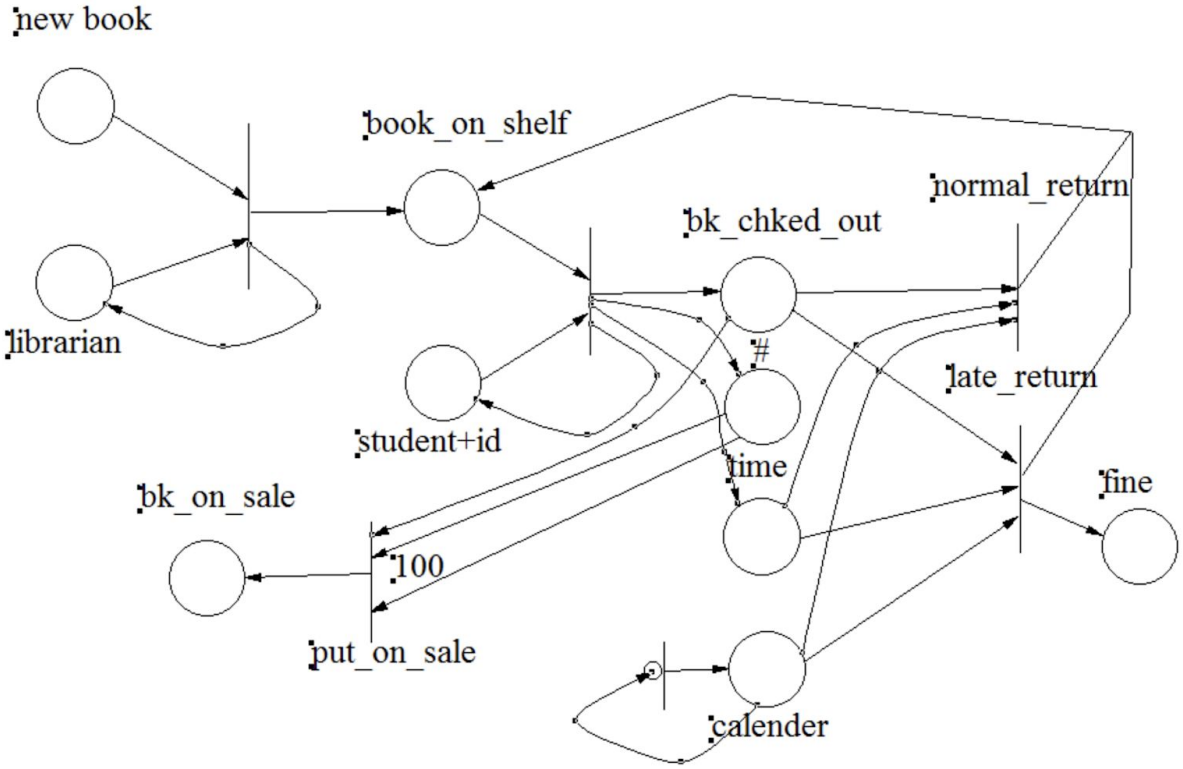
Question number 2 :
9

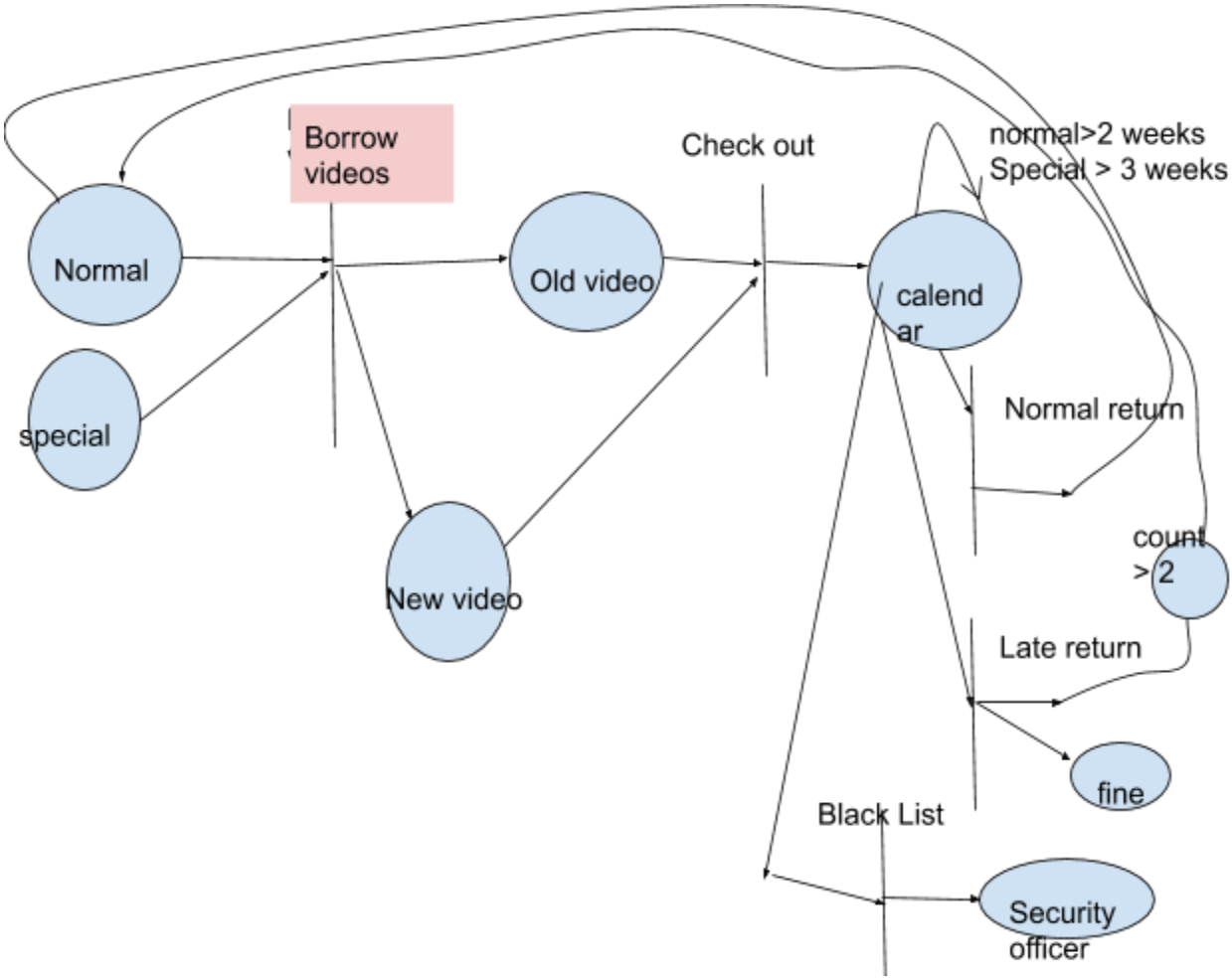



The video store has lists of entities which stores they related information and communicate within them through some flows.



Petri-Net :


new book
book_on_shelf
normal_return
bk_chked_out
librarian
#
late_return
student+id
time
fine
bk_on_sale
100
put_on_sale
calender




**JSD**

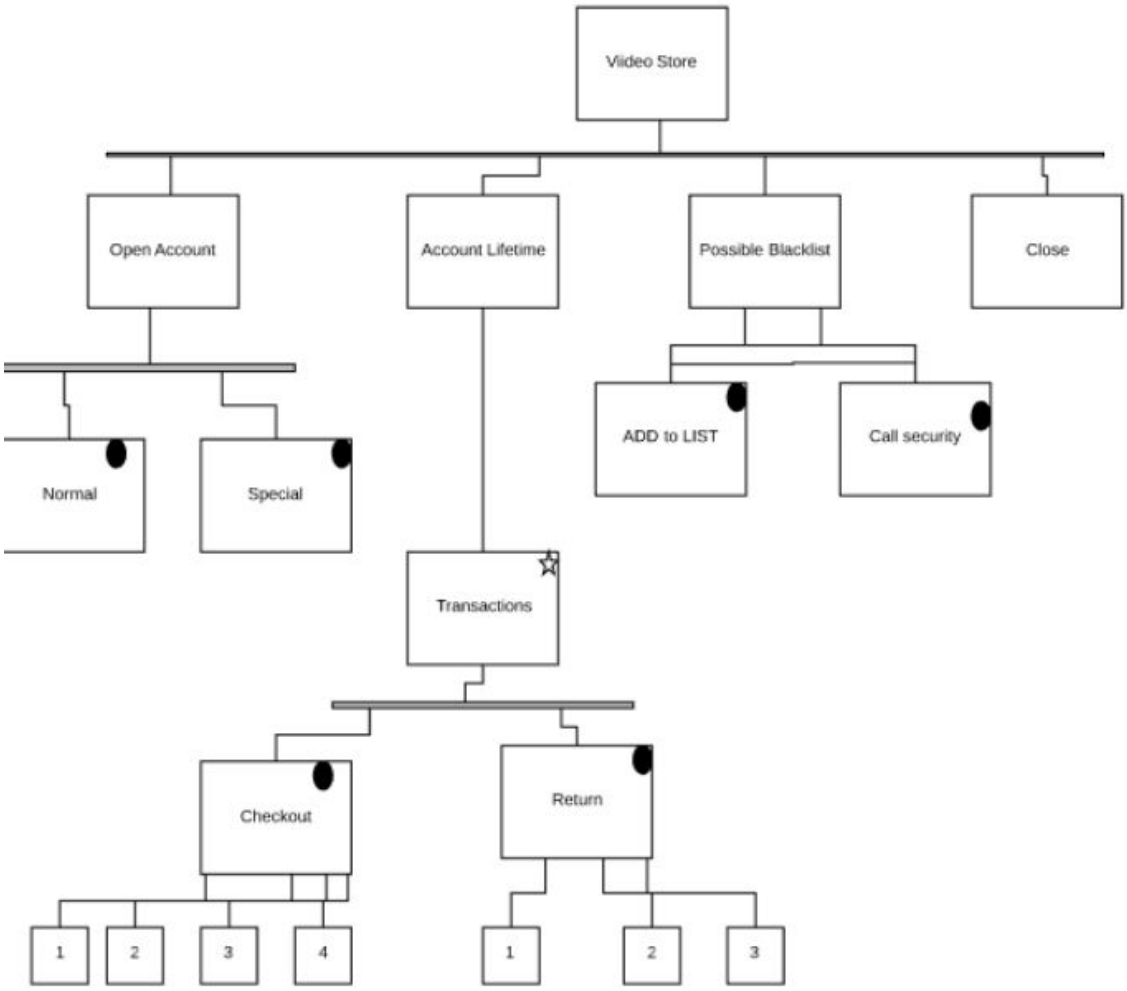
→ **the onlyyy [video](#) on JSD on youtube**

**------**

JSD tree

http://www-edlab.cs.umass.edu/~adrion/520-f04/PDF/Lecture17.pdf (notes on JSD more )

http://computing.unn.ac.uk/staff/cgpv1/downloadables/CD3005/jsd1up.pdf (could also help for JSD)




Note: the circles can be represented as ‘o’ and the star as ‘*’

Actions for checkout

---

1. Check what type of user
2. Check (if in blacklist) call security
3. Check what type of video (old or new)
4. Provide return data based on video.  Old: 2 weeks ; new : 3 days

Actions for return

---

1. Check (if late)
2. If late charge fee depending on  #of weeks late
3. Charge customer fee

Explanation:

1. Sequences: the order of the boxes indicate a sequence
   Must occur: open account ,account lifetime, possible blacklist, close
2. Selection: ‘o’ or the dark circle indicates alternatives

3. Iteration: the *  or star represent iteration. Note that an iteration of 0

4. Double lines indicate that that the boxes can occur at the same time

**Question number 3**

Coupling and Cohesion

a) Label and justify the cohesion levels of the following modules
   i) download a digital picture; play a digital video
   ii) parse the search criteria ; search the DB for matches
   iii) locate the record to be updated from DB; update the record; save the record back to DB file
   iv) design the system using OOD; implement the system using OOP

   **Definition of Coupling ->** degree of interdependence between the modules

   --Good Software ----- low coupling

   **Definition of Cohesion ->** Measure of the degree to which the element of the modules belong together
   ---> glue that keeps all  modules together
   ---> Good software -------> High cohesion

i) Download a digital picture; play a digital video

The **Coincidental cohesion** is the worst , when module grouped arbitrarily the relation is just they group together as in Download a digital picture; play a digital video tasks there are no relationship.  —- TRASHBIN

ii) Parse a search criteria ; search the DB for matches

The **Communicational cohesion** module grouped task which operate on the same data. The Parse the search criteria; search the DB for match are using the same input or output data.

iii) locate the record to be updated from DB; update the record; save the record back to DB file

The **Sequential cohesion** module grouped because output from task is the input to other task as sequence. The Locate the record to be update from DB; update the record; save the record back to the DB file perform in sequence so it is Sequential cohesion.

iv) design the system using OOD; implement the system using OOP

The **Procedural cohesion** module group because certain sequence of steps has to be carried out to gain objective. As in Design the system using OOD; implement the system using OOP, the implementation step can perform after the design step so it is Procedural cohesion.

**Types of Coupling:**

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

**Types of Cohesion:**

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

b) Label and justify the coupling relation between every pair of the following modules (global and private variables are prefixed by g and p respectively
Module a:
P.a = (module) c(0,1); if (p.a==1) g.flag=8;

Module b:
g.flag=0; p.c=0;

Module c; parameter c1, c2;
p.c=1; if(c1<c2)p.c=8; if (c1>2) count <<c1<<endl;

1. The coupling between module a and b is **common coupling** because they share the same global data.
2. The coupling between module a and c is **content coupling** because module c is occur in the module a.
3. The other coupling between module c and a is **control coupling** because the parameters passing by module a control the sequence of module c.

4) Give an integer array A of size n that is already in ascending order, for a query value v, find the five elements from A that are closest to v, that is, with 5 smallest possible |A[i] -v|. Pay attention to various special cases and don't take anything for granted. Indicate and prove the time complexity of your function.

https://www.geeksforgeeks.org/find-k-closest-elements-given-value/

```
def findCrossOver(arr, low, high, x) :

    # Base cases
    if (arr[high] <= x) : # x is greater than all
        return high

    if (arr[low] > x) : # x is smaller than all
        return low

    # Find the middle point
    mid = (low + high) // 2 # low + (high - low)// 2

    # If x is same as middle element,
    # then return mid
    if (arr[mid] <= x and arr[mid + 1] > x) :
        return mid

    # If x is greater than arr[mid], then
    # either arr[mid + 1] is ceiling of x
    # or ceiling lies in arr[mid+1...high]
    if(arr[mid] < x) :
        return findCrossOver(arr, mid + 1, high, x)

    return findCrossOver(arr, low, mid - 1, x)
```

```
def printKclosest(arr, x, k, n) :

    # Find the crossover point
    l = findCrossOver(arr, 0, n - 1, x)
    r = l + 1 # Right index to search
    count = 0 # To keep track of count of
              # elements already printed

    # If x is present in arr[], then reduce
    # left index. Assumption: all elements
    # in arr[] are distinct
    if (arr[l] == x) :
        l -= 1

    # Compare elements on left and right of crossover
    # point to find the k closest elements
    while (l >= 0 and r < n and count < k) :

        if (x - arr[l] < arr[r] - x) :
            print(arr[l], end = " ")
            l -= 1
        else :
            print(arr[r], end = " ")
            r += 1
        count += 1

    # If there are no more elements on right
    # side, then print left elements
    while (count < k and l >= 0) :
        print(arr[l], end = " ")
        l -= 1
        count += 1

    # If there are no more elements on left
```

\

```
# Driver Code
if __name__ == "__main__" :

    arr =[12, 16, 22, 30, 35, 39, 42,
              45, 48, 50, 53, 55, 56]

    n = len(arr)
    x = 35
    k = 4

    printKclosest(arr, x, 4, n)
```

5) For two arrays A and B that are already in ascending order, write a program to mingle them into a new array ordered C evenly, the size of C is varied, depending on the values of A and B: the even-indexed (the index starts from 0) elements come from A while odd-indexed elements are from B.

```
def mingle_arrays(A, B):
    C = []
    i, j = 0, 0
    if len(A) > 0:
        C.append(A[0])
        i = i + 1
    while i < len(A) and j < len(B):
        if i == 1:
            while(B[j] <= C[-1]):
                j = j + 1
            C.append(B[j])
            j = j + 1
        try:
            while(i < len(A) and A[i] <= C[-1]):
                i = i + 1
            C.append(A[i])
            i = i + 1
        except:
            pass
        try:
            while(j < len(B) and B[j] <= C[-1]):
                j = j + 1
            C.append(B[j])
            j = j + 1
        except:
            pass
    if i < len(A):
        try:
            while(A[i]
                  <= C[-1]):
                i = i + 1
            C.append(A[i])
            i = i + 1
        except:
            pass
    elif j < len(B):
        try:
            while(B[j] <= C[-1]):
                j = j + 1
            C.append(B[j])
            j = j + 1
        except:
            pass
    return C
```

**Differences between Black Box Testing vs White Box Testing:**

| BLACK BOX TESTING | WHITE BOX TESTING |
|---|---|
| It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure r the code or the program of the software. |
| It is mostly done by software testers. | It is mostly done by software developers. |
| No knowledge of implementation is needed. | Knowledge of implementation is required. |
| It can be referred as outer or external software testing. | It is the inner or the internal software testing. |
| It is functional test of the software. | It is structural test of the software. |
| This testing can be initiated on the basis of requirement specifications document. | This type of testing of software is started after detail design document. |

**Things that might be on the final:**

- ➔ Pros and Cons of Group Cohesiveness
- ➔ Module example, might ask which cohesion level and justify
- ➔ Might ask us to rewrite merge sort using loops [pseudo-code]

```
merge(left, right) {
    if left or right is empty {
        return left or right
    }
    result = []; i, j = 0, 0;
    while (len(result) < len(left and right combined)) {
        if left[i] < right[j] {   // left value is smaller
            result.append(left[i])
            i += 1
        }
        else {                    // right value is smaller
            result.append(right[j])
            j += 1
        }
        if left or right lists are out of space {   // left[i] or right[j] would be out of bounds
            result.extend(left[i:] or right[j:])
            break
        }
    }
    return result
}
```

```
mergesort(list) {
    if len(list) is 0 or 1 {
        return list     // already sorted
    }
    mid = len(list) / 2
    left = mergesort(list[everything up to mid])
    right = mergesort(list[everything from mid to the end])
    return merge(left, right)
}
```

- ➔ Different levels of coupling and cohesion
- ➔ Quicksort (Hoare, quickfind) linear time to find item
- ➔ 3NF
- ➔ Petri-net [for lib?]
- ➔ Jackson Development [JSD] (e.g. elevator control system)J

- ◆ **Modelling Phase:**

  In the modelling phase of JSD the designer creates a collection of entity structure diagrams and identifies the entities in the system, the actions they perform, the attributes of the actions and time ordering of the actions in the life of the entities.

- ◆ **Specification Phase:**

  This phase focuses on actually what is to be done? Previous phase provides the basic for this phase. An sufficient model of a time-ordered world must itself be time-ordered. Major goal is to map progress in the real world on progress in the system that models it.

- **Implementation Phase:**

  In the implementation phase JSD determines how to obtain the required functionality. Implementation way of the system is based on transformation of specification into efficient set of processes. The processes involved in it should be designed in such a manner that it would be possible to run them on available software and hardware.

➔ Any conceptual topics