5-SOFTWARE DESIGN

30 May 2022 12:04 PM

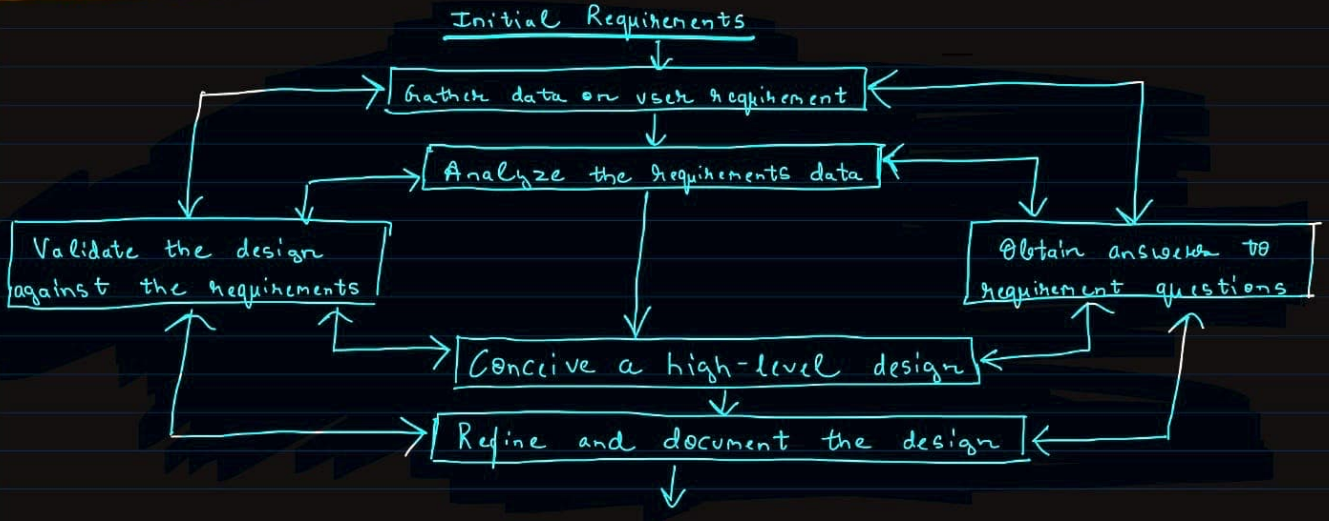
⇒ What is Design ?

⇒ A SRS document tells us "what" a system does, and becomes input to the design process, which tells us "how" a Software system works.

⇒ Designing Software System means determining how requirements are realized and result is a Software Design Document (SDD).

⇒ Thus, the purpose of design phase is to produce a solution to a problem in SRS document.

⇒ Design Framework

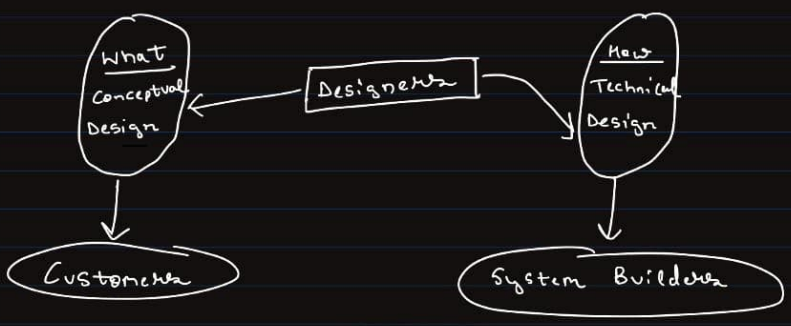


⇒ Conceptual and Technical Designs

⇒ For the reason that the designers need to satisfy both the customers and the system builders, design is really a two part, iterative process.

⇒ First, a conceptual design is produced that tells the customer exactly what the system will do

⇒ once it gets approved, it gets translated into a much more detailed document, the technical design, that allows system builders to understand the actual hardware and software needed.



A two part design process

Customers → System Builders

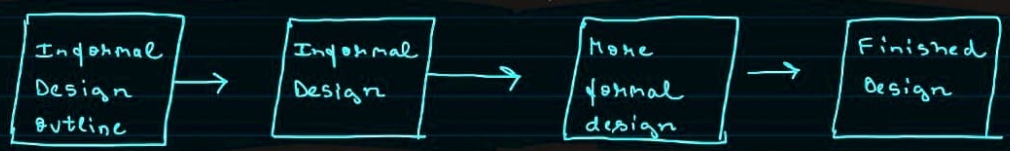
A two part design process

⇒ The conceptual design describes the system in language understandable to the customer. It does not contain any technical jargons and is independent of implementation.

⇒ The technical design describes the hardware configuration, the software needs, the communications interfaces, the input and output of the system, the network architecture, and anything else that translates the requirements into a solution to the customer's problem.

⇒ Objectives of Design

The design needs to be
- Correct and complete
- understandable
- At the right level
- Maintainable, and to facilitate maintainance of the produced code.



Transformation of an informal design to a detailed design

⇒ Why Design is Important ?

⇒ Software design should contain a sufficiently complete, accurate and precise solution to a problem in order to ensure its quality implementation.

⇒ There are three characteristics that serve as a guide for the evolution of a good design :-

1) The design must implement all of the explicit requirements contained in the analysis model and it must accomodate all of the implicit requirements desired by the customers.

2) - The design must be readable, understandable guide for those who generate code and those who test and subsequently support the software.

3) - The design should provide a complete picture of the software, addressing the data functional and behavioural domain from an implementation perspective.

unctional and behavioural domain from an implementation perspective.

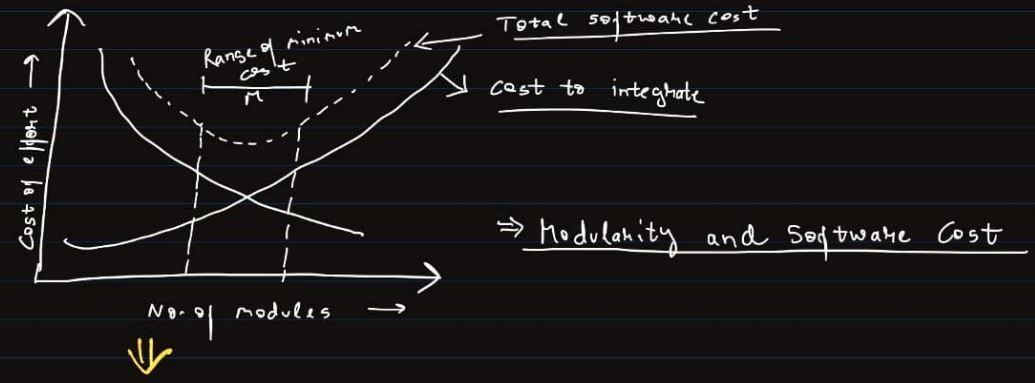
⇒ Modularity

A modular system consists of well defined, manageable units with well defined interfaces among the units.

⇒ Properties [WS EC S V]

1)- Each module is a well-defined subsystem that is potentially useful in other applications.
2)- Each module has a single well defined purpose.
3)- Modules can be separately combined and stored in library.
4)- Modules can use other modules
5)- Modules should be easier to use than to build.
6)- Modules should be simpler from the outside than from the inside.

⇒ To what extent we should modularize ?

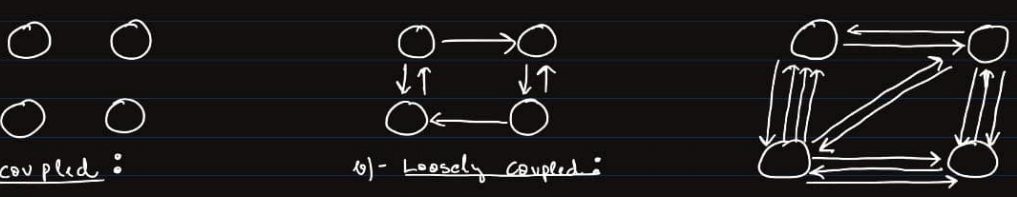


⇒ Modularity and Software Cost

Under modularity and over modularity in a software should be avoided.

⇒ Module Coupling

⇒ coupling is the measure of the degree of interdependence between modules.
⇒ Loosely coupled systems are made up of modules which are relatively independent.
⇒ Highly coupled systems share a great deal of dependence between modules.



a) Uncoupled : no dependencies

b)- Loosely coupled : some dependencies

c)- Highly coupled : many dependencies.

⇒ A good design will have low coupling. Thus interfaces should be carefully specified in order to keep low value of coupling.

⇒ Coupling is measured by the no. of interconnection between modules.

→ Coupling is measured by the no. of interconnection between modules.

Types of Coupling [DSCECC] → dyal singh college me economy class class

→ Data → Stamp → Control → External → Common → Content
(Best) (Worst)

1)- Data Coupling (Best)
- The dependency b/w modules is said to be data coupled if it is based on the fact they communicate by only passing of data
- other than that, modules are 'independent'.
- Thamp data shouldn't be there in modules.

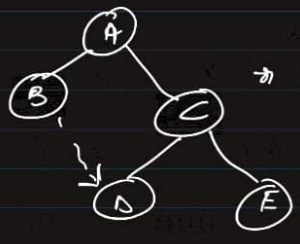
2)- Stamp Coupling
- When complete data structure is passed from one module to other
- Stamp coupling involves thamp data

3)- Control Coupling
- Two modules are said to be control coupled if they communicate by passing of control info.
- It is accompanied by means of flags that are set by one module and reacted upon by the dependent module.

4)- External Coupling
- A form of coupling in which a module has a dependency to other module, external to the software being developed or to a particular type of hardware.
- This is basically related to the communication to external tools and devices.

5)- Common Coupling
- With common coupling, module A and B have shared data.
- Making change to the common data, means tracing back to all the modules which access that data to evaluate the effect of change.
- It gets difficult to determine the module responsible for changes.

6)- Content Coupling
- Content coupling occurs when Module A changes data of Module B or when control is passed from one module to the middle of another.



→ Module B branches into D, even the D is supposed to be under control of C.

⇒ Module Cohesion

⇒ Cohesion is a measure of the degree to which the elements of a module are functionally related.
⇒ Cohesion may be viewed as a glue that keeps the module together.
⇒ It is a measure of the mutual affinity of the components of a module.

⇒ Design Objective: Maximize the module cohesion and minimize the module coupling.

Types of Cohesion [FSC PTLC] ⇒ FS pe CP gye TLC dekha

Functional → Sequential → Communicational → Procedural → Temporal → Logical → Coincidental
(Best) (Worst)

1)- Functional Cohesion
- X and Y are a part of single functional task. This is a very good reason for them to be contained in the same procedure
- Such a module often transformed a single input datum into a single output datum.
- Eg:- calculate current GPA or cumulative GPA

2)- Sequential Cohesion
- X outputs some data which forms the input to Y. This is a reason for them to be contained in the same procedure.
- A component is made of parts that need to communicate / exchange data from one source for different functional purposes.
- Eg:- addition of marks to calculate GPA, input for preparing result.

3)- Communicational Cohesion
X and Y both operate on the same input data or contribute towards the same output data. This is okay, but we might consider making them separate procedures.

4)- Procedural Cohesion
⇒ X and Y are structured in same way.
⇒ Poor reason for putting them in same procedure.
⇒ occurs in modules whose instructions although accomplish different tasks yet have been combined because there is a specific order in which tasks are to be completed.
⇒ Eg:- "Calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA" is case of procedural cohesion
⇒ Since these modules consist of instructions that accomplish several tasks that are virtually unrelated, these types of module tend to be less maintainable.

5)- Temporal Cohesion
⇒ X and Y both must perform around same time.
⇒ Module exhibits temporal cohesion when it contains task that are related by the fact that all tasks must be executed in the same time-s
⇒ **Not a good** reason to put them in same procedure.

6)- Logical Cohesion
- X and Y perform logically similar operations.
- It occurs in the modules that contain instructions that appear to be related because they fall into the same logical class of functions.

7)- Coincidental Cohesion
- X and Y have no conceptual relationship other than shared code.
- Exists in modules that contain instructions that have little or no relationship to one another.
- That is, instead of creating two components, each of one part, one one component is made with two unrelated parts
- Eg:- check validity and print is a single component with two parts.
- To be avoided as far as possible.

⇒ Relationship b/w Cohesion and Coupling

be avoided as far as possible.

⇒ Relationship b/w Cohesion and Coupling

⇒ A good software design processes clean decomposition of a problem into modules and the arrangement of these modules in a neat hierarchy.
⇒ Therefore, a software engineer must design the modules with goal of high cohesion and low coupling.
⇒ Ex - 'plug and play' feature of Computer system.

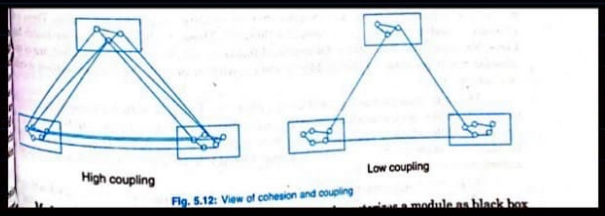


Fig. 5.12: View of cohesion and coupling

⇒ Strategy of Design

⇒ A good system design strategy is to organize the problem modules in such a way that are easy to develop, and later to, change.

Why it is important ?

1)- Even the existing code, if any, needs to be understood, organized and pieced together.
2)- It is still common for the project team to write some code and produce original programs that support the application logic of the system.

⇒ Many notations other than flowcharts, have been designed for expressing designs, that help in minimizing the length of jumps from specifications to design and design to code.

⇒ Bottom - Up design

⇒ Common Approach is to identify modules that are required by many programs.
⇒ These modules are collected together in the form of "library".
⇒ In order to provide largest modules from the existing ones, a cross-linked tree structure is built, in which each module is subordinate to those in which it is used.

⇒ Weakness : A lot of intitution would be used to decide what functionality a module should provide.
⇒ If a system is to be built from an existing system, this approach is more suitable, as it starts from some existing modules.

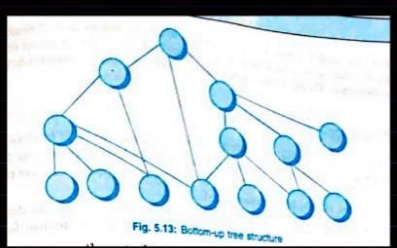


Fig. 5.13: Bottom-up tree structure

Fig. 5.13: Bottom-up tree structure

⇒ Top down Design

⇒ It's approach start by identifying the major modules of the system, decomposing them into lower level modules and iterating until the desired level of detail is achieved.

⇒ It is suitable if the specifications are clear and developement is from the scratch.
⇒ If coding of a part starts soon after its design, nothing can be tested until all its subordinate modules are coded.

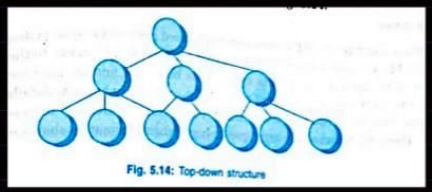


Fig. 5.14: Top-down structure

⇒ Hybrid Design

⇒ Pure top-down or pure bottom-up approach are often not practical.
⇒ For a bottom-up approach to be successful, there must be a good notion at the top to which the design should be heading.

⇒ For a top-down approach to be effective, some bottom-up approach is essential for following reasons :
1)- To permit common sub modules
2)- Near the bottom of the hierarchy, where the intuition is simple, and the need for bottom up testing is because there are more no. of modules at low levels than at high levels.

3)- In the use of the written library modules, in particular, reuse of modules.

⇒ Hybrid approach has become popular after acceptance of reusability of modules.
⇒ Microsoft Foundation classes (MFCs), object-oriented concepts are steps in this direction.

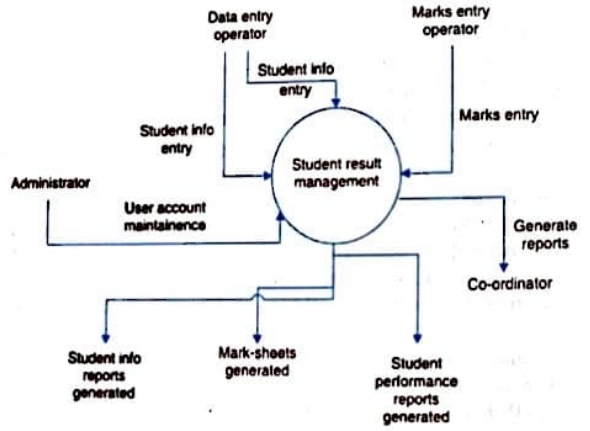
⇒ Function Oriented Design

Every subject has some credit points assig... >= 50 in a subject, he/she is considered 'Pass' in that subject otherwise the student is considered 'Fail' in that subject. If a student passes in a subject, he/she earns all the credit points assigned to that subject, but if the students fails in a subject he/she does not earn any credit points in that subject. At any time, the latest information about subjects being offered in various semesters and their credit points can be obtained from university's website.

It is required to develop a system that will manage information about subjects offered in various semesters, students enrolled in various semesters, elective (s) opted by various students in different semesters, marks and credit points obtained by students in different semesters. The system should also have the ability to generate printable mark-sheets for each student. Semester-wise detailed mark lists and student performance reports also need to be generated.

3.9.2 Context Diagram

The context diagram is given below:



The following persons are interacting with the "student result management system"

(*i*) Administrator

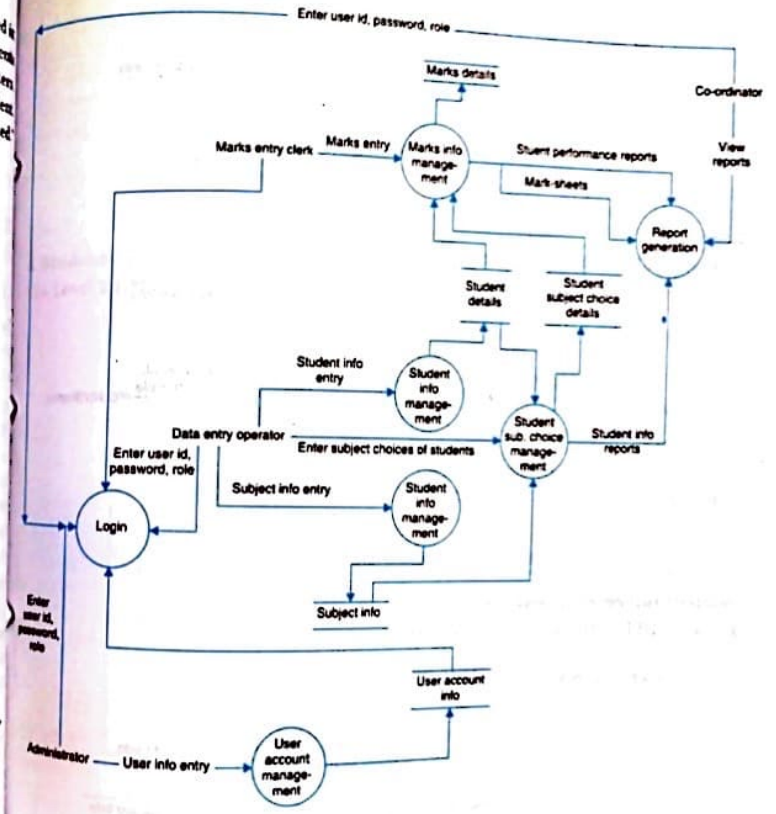
(*ii*) Marks entry operator

(*iii*) Data entry operator

(*iv*) Co-ordinator

Level-1 DFD

The Level-1 DFD is given below:



Level 1 DFD of result management system