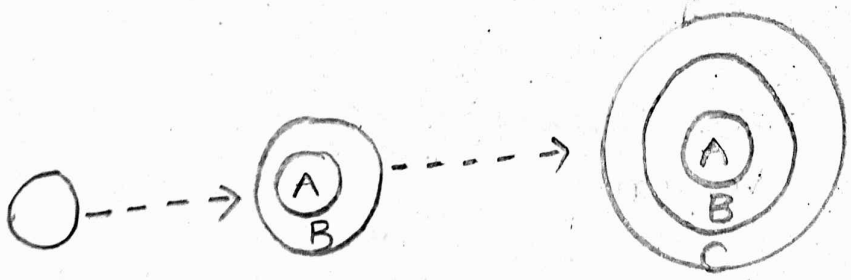
Assignment - 1

1. Draw and Explain Incremental process model

A) Incremental process Model:

This life cycle model is sometimes referred to as successive versions model & incremental model. In this model first a simple working system with basic features is built and delivered. over the iterations, successive versions are implemented & delivered to the customer until desired system is realised.

In this model the requirements of software are broken down into several modules / features for easy incrementation & development, so that no long-term plans are made. Hence it is easier to accomodate change requests from the users / customers.



A, B, C are modules of a software product that are incrementally developed and delivered.

After the requirements gathering & specification, it is split into several versions stating from core.

In Each successful increment, the next version is constructed using iterative waterfall model. After the last increment the full version of Software is developed.

Let us see the Diagrammatic Representation of Incremental model of Software Development.

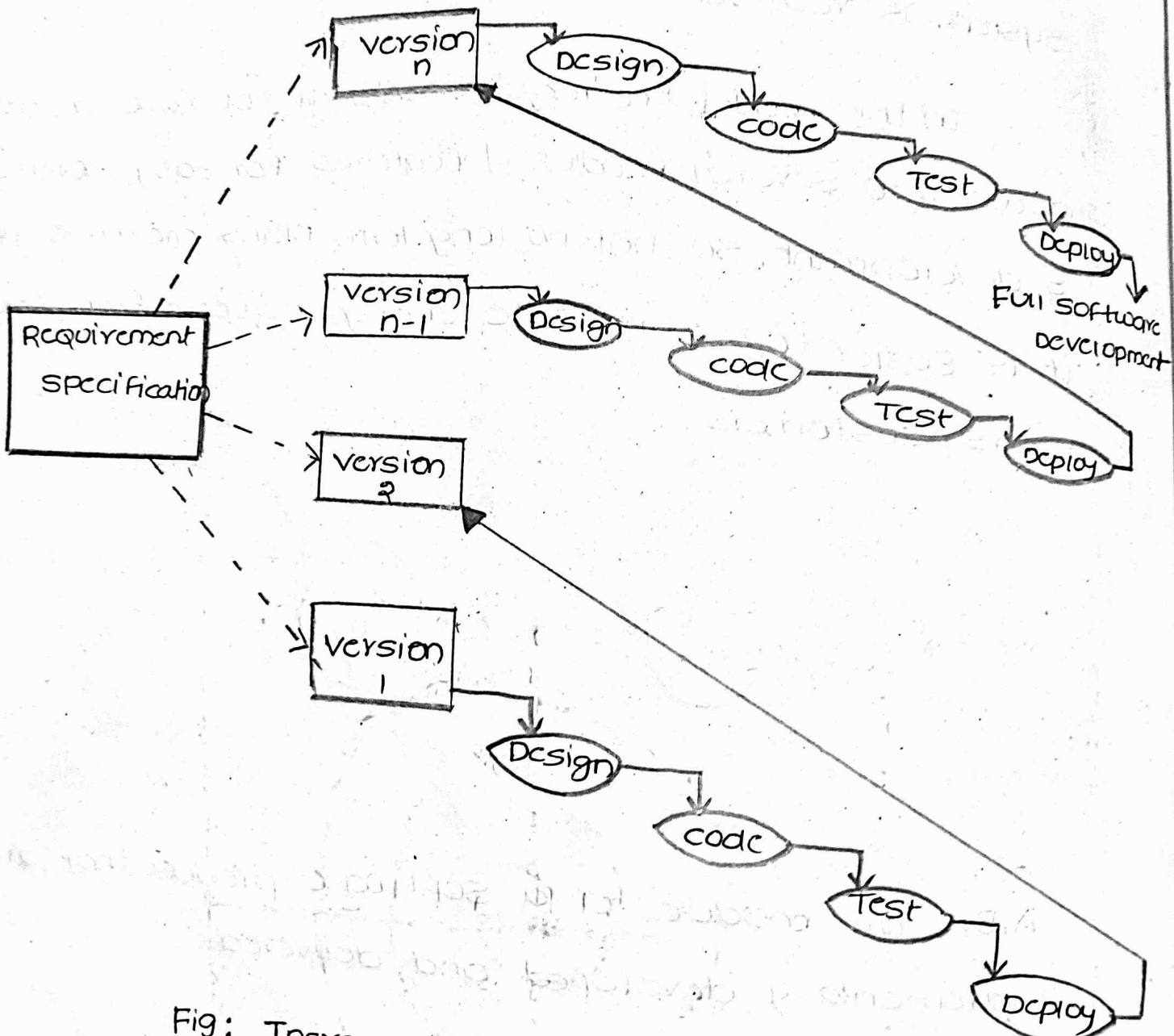


Fig: Incremental Model of Software Development

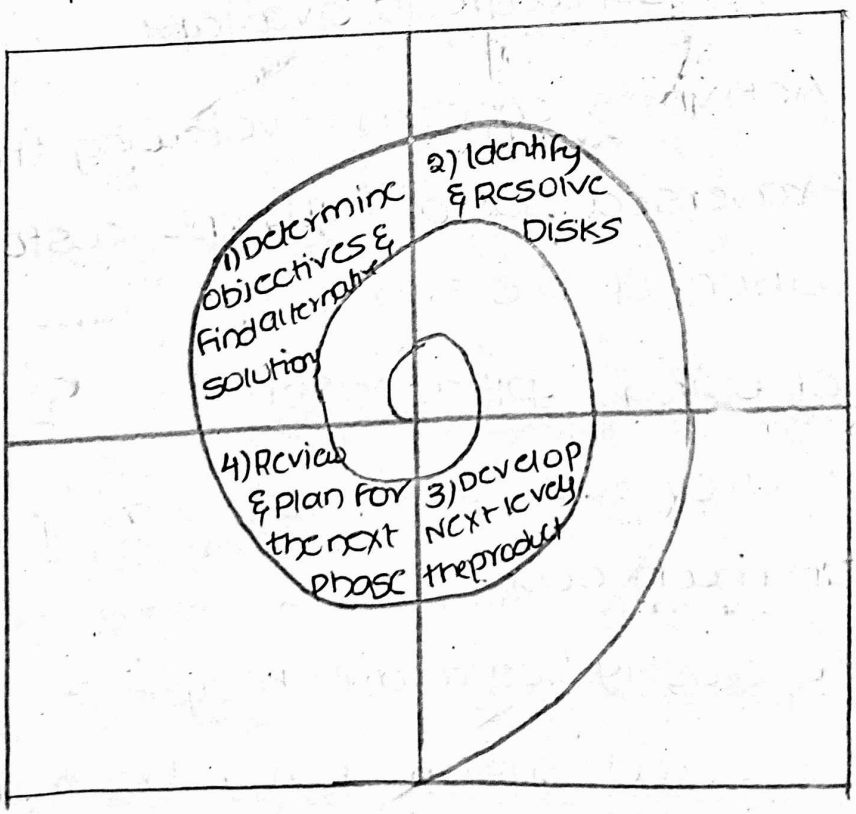
2. Write about spiral Model?

SPIRAL MODEL:

This model looks like a spiral, with loops. The Exact no of loops of the model is not fixed & can vary based upon the project. Each loop of the spiral is called "phase of the software process.

The Exact number of phases through which the product is developed is based on the project manager & manager project risk. The main feature of spiral model is to handle unforseen risks that show up after the project has stated. In this model the prototy -pes are build at the start of every phase.

Over each loop one or more features are elaborated analysed & risks are identified & resolved.



Phases of spiral Model

Each phase is split into four sectors or quadrants. In the first quadrant few features of software are identified for immediate development with each iteration around the spiral, progressively more complete versions of software are build.

Quadrant 1: The objectives are investigated, elaborated, analysed. Based on this the risks are identified. In this quadrant alternative solutions possible are considered.

Quadrant 2: In this alternative solution are evaluated, to select the best solution. This is done by developing appropriate prototype.

Quadrant 3: Developing & verifying the next level of software. At the end, the identified features have been implemented & next version of software is available.

Quadrant 4: Activities concern reveicwing the results of the stages traversed so far with the customer & planning the next iteration of the spiral.

Advantages of using Spiral Model

* This model provides early indication of existence of risks, without much cost.
* users can be closely tied to all lifecycle steps.
* customers see the system early because of rapid prototyping tools

* Early & Frequent feedback.

Disadvantages of Spiral Model

* The Model is complex.

* Risk assessment requires Expertise.

* Time spent planning, doing risk analysis & prototyping may be exercised.

* Time spent for risks evaluation is too large for small / low risk projects.

3. What is Agility? Explain about Etreme programming.

Ans: Agility is defined as

- Effective to change
- Effective communication
- Drawing customers on to team
- organizing a team so that it is in control.

Agility can be applied to any software process. The process includes

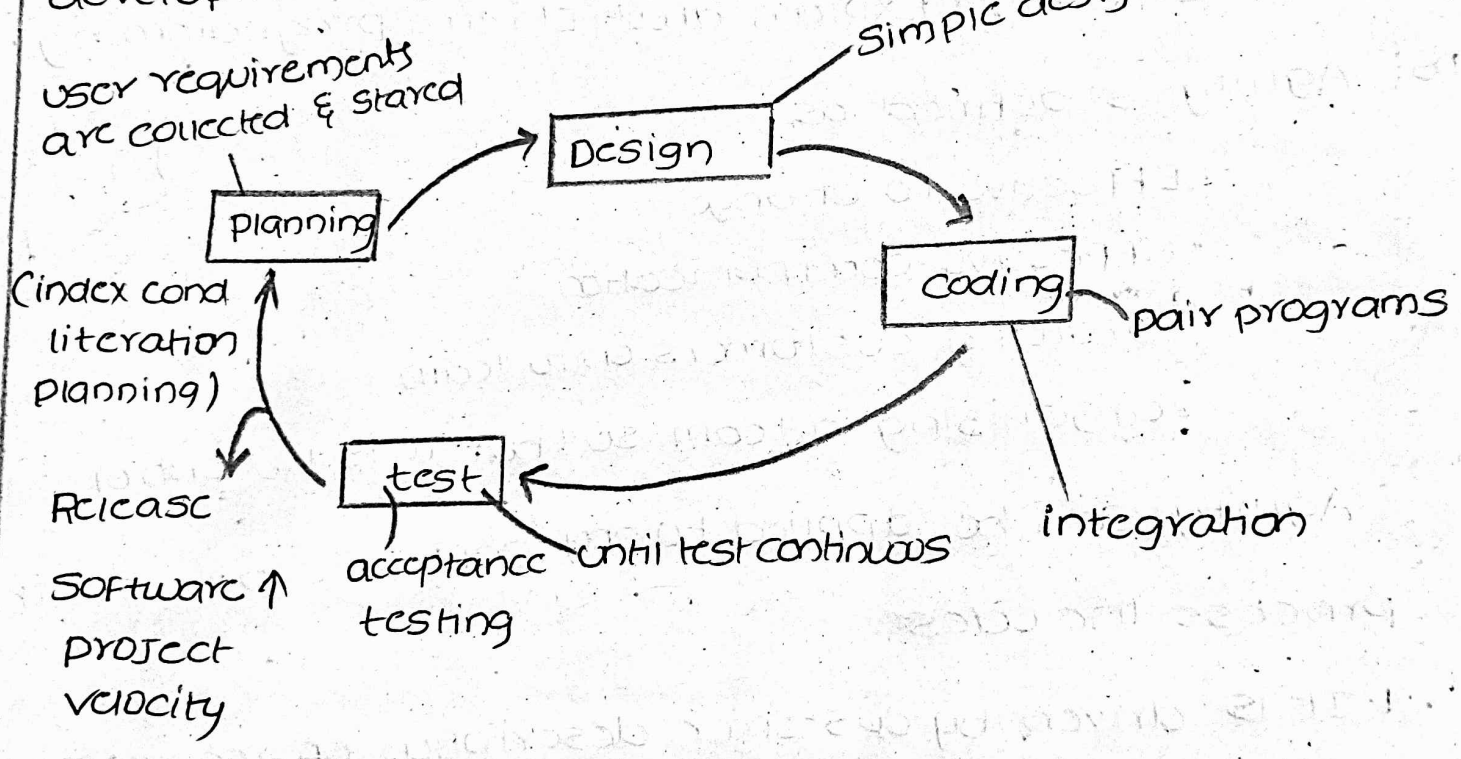
1. It is driven by customer description of requirements
2. Recognize that plans are short lived.
3. Develops software iteratively
4. Delivery multiple software increments
5. Adopts as changes occur

Extreme programming (XP):

Extreme programming is an agile software development method. XP is a light weight, efficient, low risk, flexible & fun way to develop a software small to medium sized that work to change requirement.

Goals:

* To produce high quality software.
* To reduce the cost of changes by having multiple short development cycles.



EXTREME PROGRAMMING PROCESS

The XP process: It has 4 framworks

a. XP planning

Begins with set of requirements, placed on index card. A priority is assigned, The agile team assign cost

The reg are grouped for a delivered increment commited on delivery date.

b. XP Design

Follows KIS (keep It Simple) principle. Encourage, use of CRC code, Refactoring. Design occurs both before & after coding commences.

c. XP coding

Recommends construction of unit tests Encourages pair programming Needs continuous interaction with other portions of software, that provides "smoke testing" Environment.

d. XP - Testing

Unit test should be implemented using Frame Works Integration & validation Testing occurs on daily basis. Acceptance Tests also called "Customer tests' are specified by customers & Executed.

They are derived from user Requirements.

4. Explain about characteristics of a Good SRS Document?

Following are the characteristics of a good SRS document.

1. correctness

User review is used to ensure the correctness of requirements Stated in SRS. SRS is said to be connect

If it covers all the requirements that are actually expected From the system.

2) completeness

Completeness of SRS indicates every sense of completion including the numbering of all pages, essential req., Performance, design, constraints & attributes.

3) Consistency

Requirements in SRS are said to be consistent if there are no conflicts blw any set of requirements. There are 3 types of possible conflicts

* The specified characteristics of real-world objects.
* Reasonable/temporal conflict blw 2 specified actions.
* Two/more requirements may define the some-real world object but use different terms.

4) Unambiguousness

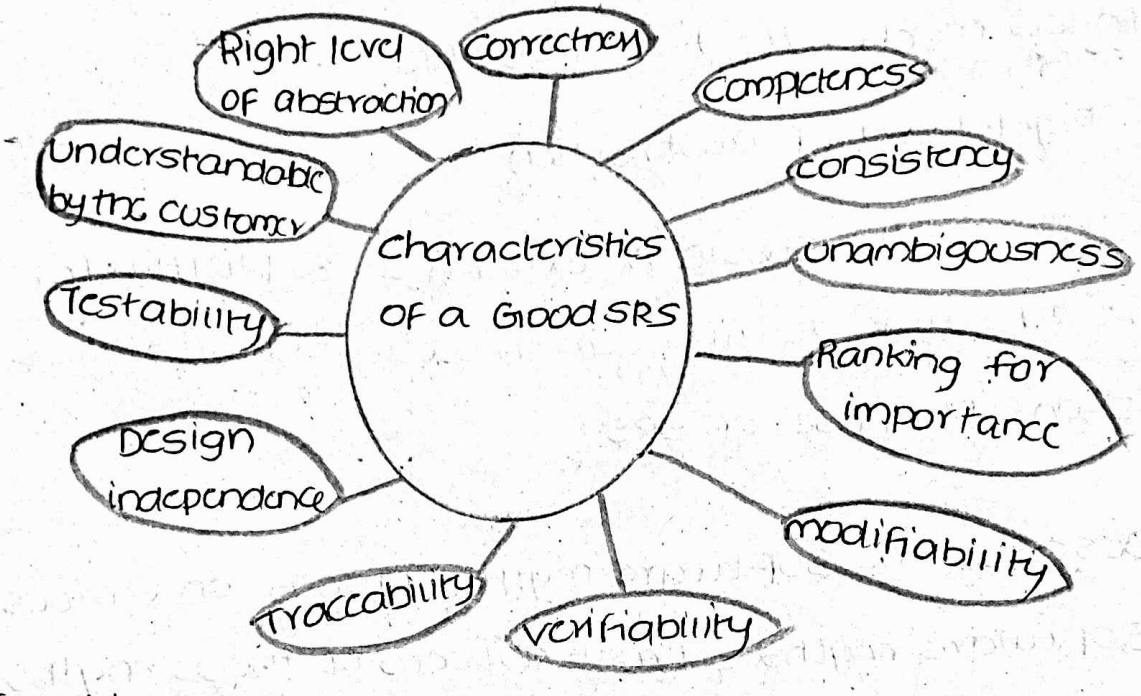
When every fixed requirement how only 1 interpretation. It is clear and simple to understand.

5) Ranking for importance & Stability

If each requirement in it has an identifier to indicate either the significance/stability. Another way i.e to distinguish classes of items as Essential conditional optional.

6) Modifiability

It should be made as modifiable as possible & should be capable of easily accepting changes. they should be properly indexed



7) Verifiability

If there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system

8) Traceability

If the origin of each requirements is clear. It is of 2 types.

1) Backward - depends upon req. Explicitly

2) Forward - depends upon req Element in SRS having a unique name / reference number.

9) Design Independence

To choose from multiple design alternatives for the final

System.

10. Testability

Written in such a way that it is easy to generate test cases & test plans.

11. Understandable by the customer:

The use of formal notations & symbols should be avoided to as much extent as possible.

12. Right Level of abstraction

The details should be explained explicitly. for feasibility study fewer details can be used. Hence it values according to the purpose of SRS.

5. Describe the software myths? Discuss on various types of software myths & trust aspects of these myths?

Ans Software myths - beliefs about software & the process used to build it myths appear to be reasonable state of fact.

Types of Myths:

1. Management myths - Manages with software responsibility. Often grasps at belief in a software myth.

Myth: We already have a book that's full of standards & procedures for building software.

Reality: The book & standards may visit but is it used? is it complete? is it adaptable? Does it reflect modern Softwar

engineering practise?

Myth: If we get behind schedule we can add more programmers and catch up

Reality: SD is not mechanistic process like manufacturing However an new people are added, time must be spent guiding members, rather than developing.

Myth: If we decide to outsource the software project to a third party I can just relax & let that firm built it.

Reality: If an organization does not able to manage control it will invaiably struggle when it out sources software project.

2. customer myths: It leads to false Expectations & ultimately dissatisfaction

Myth: A general state of Q objectives is sufficient to begin writting programs

Reality: A poor up-front definition is major cause of failed software efforts A format & detailed description Q functions, behaviour & validation criteria is exential.

Myth: project request continuosly change but change can be easily accomodated because software is flexible.

Reality: The impact of software req change vaies with the time at which it is introduced The cost impact grows rapidly. Resources have been committed change when requested after

Software is in production can be much Expensive.

3. Practitioner's myths:

Myth: once we write the program & get it to work, our job is done.

Reality: Industry data indicate that between 60 & 80% all effort Expanded on software will be done after it is delivered to the customer for the first time.

Myth: until I get the program "running" I have no way of accessing its quality.

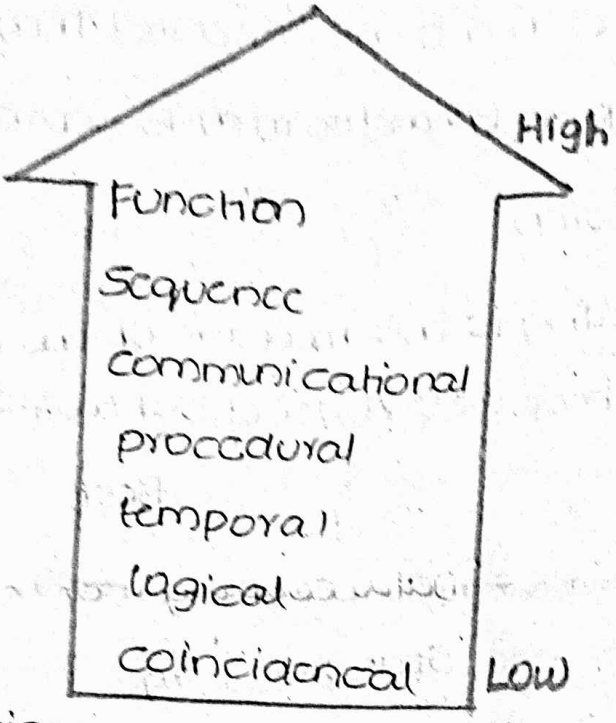
Reality: SE quality can be applied from the inception of a project - the formal technical review.

Assignment-2

1. Explain about cohesion & coupling?

Ans COHESION: It is a measure of the degree of which the elements of the module are functionally related Basically cohesion is the internal glue that keeps the module togeth

A good software design will have high cohesion



TYPES OF cohesion:

* Functional cohesion: It performs the task & functions. It is an ideal situation.

* Sequential: An Element outputs some data that becomes I/P for other. It occur Naturally in Functional programming Language.

* Communicational: Two elements operate on same input data/ contribute towards same O/P data.

Ex: update record & send it to printer

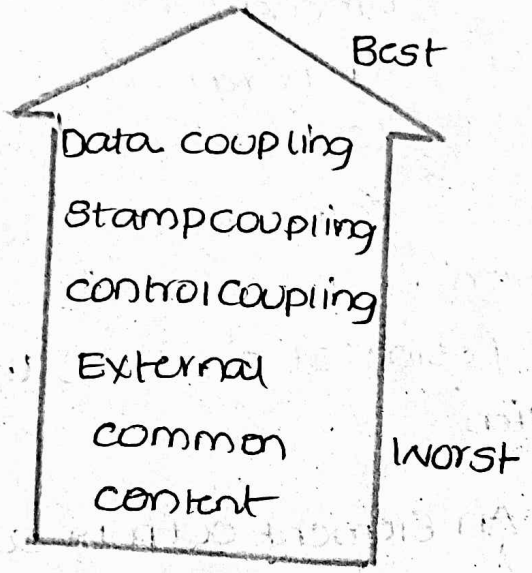
* proccdural: Elements ensure the order of execution. Actions are weakly connected & unlikely to be reusable.

* Temporal: elements are related by their timing. A module connected with temporal cohesion all the tasks must be executed in the same time but of different activities occur at unit time.

* Logical: the elements are logically related not functionally.

* coincidental: elements are unrelated they have no conceptual relationship other than location is source code. It is worst form of cohesion.

Coupling: coupling is the measure of the degree of interpendence b/w the modules. A good software will have low coupling.



Types of coupling

* Data coupling: If the dependency b/w modules is only through passing data then they are in data coupled the component are independent.

* Stamp: The complete data structure is passed from one module to another module Hence it involves tramp data

*control: If modules communicate by passing control information, it can be used bad IFF parameters indicate completely different behaviour.

*Extended: In Extended the modules depend on other modules external to software being developed or to a particular type of hardware.

*common: The modules shared data such as global data. The change in global data means tracking back to still all modules. so it has difficulty in reusing modules etc..

*content: In a content coupling one module can modify the data of another module or control flow is passed from one module to other. this is the worst from of coupli & should be avoided.

---

2. Explain about White Box testing ?

Ans: White-Box Testing

White-Box testing is an important type of unit testing. A Large no. of strategies Exist. each strategy designs test cases based on analysis of source code. It can be Either coverage-based or Fault-based.

Fault based testing

A Fault based testing strategy targets to detect certain types of faults. These faults focuses on consti

the fault model of strategy.

Ex: mutation strategy

Coverage-based testing

A coverage based testing strategy attempts to execute certain elements of a program.

Ex: statement branch, multiple coverage etc.

Testing criterian for coverage-based Testing.

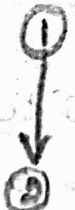
The set of specific prog. elements that a testing strategy targets to execute is called the testing criterion of the strategy.

We can compare two testing strategies by detecting whether one is stronger, weaker or complementary to other.

A white box testing strategies is said to be stronger testing strategy covers all program else covered by weaker testing strategy & it additionally covers at least one prog. element that is not covered by weaker strategy

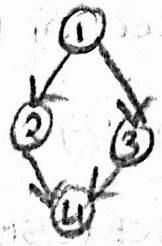
Sequence

```
1. a=5;
2. b=a*2-1
```



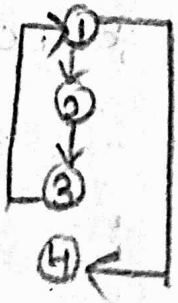
Selection.

```
1. IF(a>b)
2. c=3;
3. else c=5;
4. c=c*c;
```



Iteration.

```
1. while (a>b{
2. b=b-1
3. b=b*a 3.}
4. c=a+b;
```



3. Explain about testing object oriented programs?

Ans: Object orientation would reduce the cost of & Effective incurred on testing. It is based on feature such as Encapsulation, abstraction, reuse through Inheritance.

Procedures are the basic unit of testing. Since methods is an object oriented program are analogous to procedure is a procedural program.

All the methods share the data of the class. A method has to be tested will all the other methods & data of the corresponding object. Hence a method can't be considered as basic unit of testing of OOP.

Encapsulation helps in data abstraction, error isolation error prevention It prevents tester from accessing data thus it makes testing difficult.

Inheritance helps in code reuse & was expected to simplify testing.

Dynamic binding makes the code compact elegant & Easily Extensible. For testing it is not Easy since the binaings takes place at run time.

Object states: behaviour of object in different in differe States. Hence testing an object in only one of its states is not Enough. It has to be tested in all states.

Gray-Box testing of object-oriented programs

For oop several types of test cases can be designed based on the design models of oop. These are called the gray-box test cases. The following are some imp types of gray-box testing that can be carried on based on UML models:

State model based:

. Each method of any object are tested all Each State & obj.

State transition coverage: Whether all transactions depend work Satisfactorily.

Path coverage: All Paths in model are tested.

Use case based:

Scenario based: consists of a mainline scenario & several alternate scenarios they both are tested to check if any Errors shows up.

Class diagram based:

Testing derived classes: All derived classes of the base class have to instantiated & tested.

Associative testing: Associative relations are tested.

Aggregation testing: Aggregate objects are created test

Sequence diagram based:

method coverage: All methods depicted in seq diagrams are covered

Integration Testing of object oriented programme

Two main approaches

1) Thread based

2) use based

Thread based: All classes that need to colloborate to realis the behaviour of a single use case are integrated & tested After that another use case is taken up.

User based: It begins by testing classes that either need no service from other classes or need services from at most a few other classes. After they have been tested, classes that use the services. From already integrated classes are integrated & tested.

4. Describe SEI capability maturity model (CMM)?

SEI capability maturity model (CMM):

CMM was proposed in 1987 by Software Engineering institute (SEI). It is a framework that is used to analyse the approach & techniques followed to develop Software product. It also provides guideline It helps to improve quality of software development into that following five maturity levels.

Level 1: Initial

The software develop at this level is characterised by adhoc activities very few or no processes are defined Hence they follow own processes so development efforts become chaotic :- It is also called "chaotic level".

Level 2: Repeatable

In this Level, the processes for both management is development. Such as tracking cost etc. configuration management tools are used size & cost estimation techniques like COCOMO etc are used. In this organization the successful compiction depends on team members.

Level 3: Defined

At this level, the processes for both management & development activities are defined & documented. The capabilities of Employees are build up review techniques are Emphasized.

Level 4: Managed

At this level Level the focus is on software metrics. Both process & product metrics are collected Quantitative Quality goals are set for the product at the time of

completion of development, it was checked. Thus, the result of process measurements are used to evaluate project performance rather than improving process

Level 5: Optimising

At this stage, process & product metrics are collected. Process & product measurement. The organized would identify the best software engineering practises & innovations.

They have a dept. whose sole responsibility is to assimilate latest tools & technologies & propagate them organization wide. SEI CMM provides a list of key area on which to focus to take an organisation from one level of maturity to the next. Thus it provides a way for gradual quality improvement over several stages.

CMM Levels	Focus	Key process Area (KPA)
Initial	competent people	Software project planning
Repeatable	Project Management	Software configuration management.
Defined	Definition & processes	process def, training prog, peer reviews
managed	product & process quality	Quantitative process metrics, quality Management.
optimising	continuous process improvement	Defect prevention process engagement Technology change Management.

5. Explain about CASE environment?

Ans

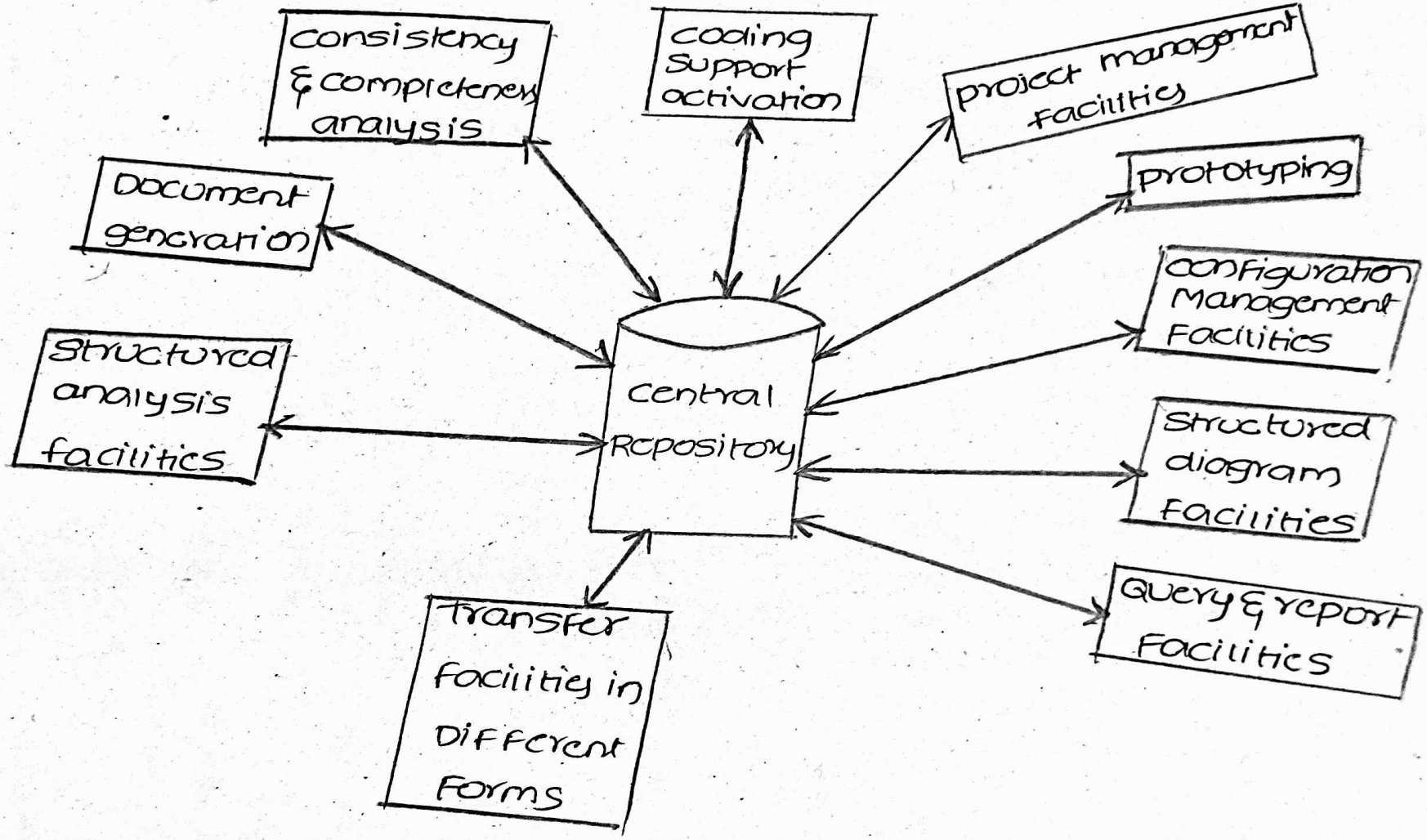
CASE Environment

The true power of a tool set can be realized only when these set of tools are integrated into a common framework or environment. If the different case tools are not integrated, then the data generated by one tool would have to input to the others tools Also many tools doesn't allow Exporting data.

Since different tools covering different stages share common information, they must be integrated CASE Environment share common information among themselves. Thus it facilitates the automation of the Step-by-step methodologies for software development.

The tools commonly integrated in a program Environment are a test editor, compiler, and a debugger. The different tools are integrated to the extent that once the compiler detects an error, the editor takes automatically goes to the statement in error & the error statements are highlighted.

[This central repository is usually a data dictionary containing definition of all composite & elementary datalem



A CASE ENVIRONMENT