q1 : Evolutionary Development, Spiral and RAD software life cycle model based on characteristics of requirements.

Answer 1 :

we are going to compare model on the basic of

1. Correctness
2. Completeness
3. Consistency
4. Unambiguosness
5. Ranking of importance and stabilty
6. Modifiability
7. Verifiablity
8. Traceabilty
9. Design dependency
10. Testability
11. Underestandable by customer
12. Right level of abstraction

1. WaterFall Model :
   - It is easy to use and used in small projects.
   - It require planning in early stage.
   - Detailed documents are required.
   - Maintenace is least.
   - Felixiblity to change is very difficult.
   - User involement is least only required in the begining of project
   - Testing is possible afer completion of coding phase.
   - Design dependency is very high.
   - Re-usabllity is very low.
   - Time frame is very high.
   - Cost is low as compare to other models.
   - Hard for customer to understand in begining.
   - It not useable in real life projects
2. Iterative Enhancement Model:
   - This model is the same as the Waterfall model but with some restrictions.

Generally it provide feedback paths from every phase to its preceding phases,
Which is the main difference from the classical waterfall model.
The feedback allow use to detect and correcting the errors which we has committed during the some phase.

3. Evolutionary model :

It is combination of iterative and incremental model of software development life cycle. It divides the development cycle into smaller, it is helpful since it give access to product

after every cycle. So error and bugs can be found. It is used for large projects so that no big bug will be reported after the completion of the softwares.

4. Spiral Model :

It is mainly used to find the risk in the model. This model look like spiral with many loops. Risk handling is main work of this model.

5. RAD model(Rapid Application Development) :

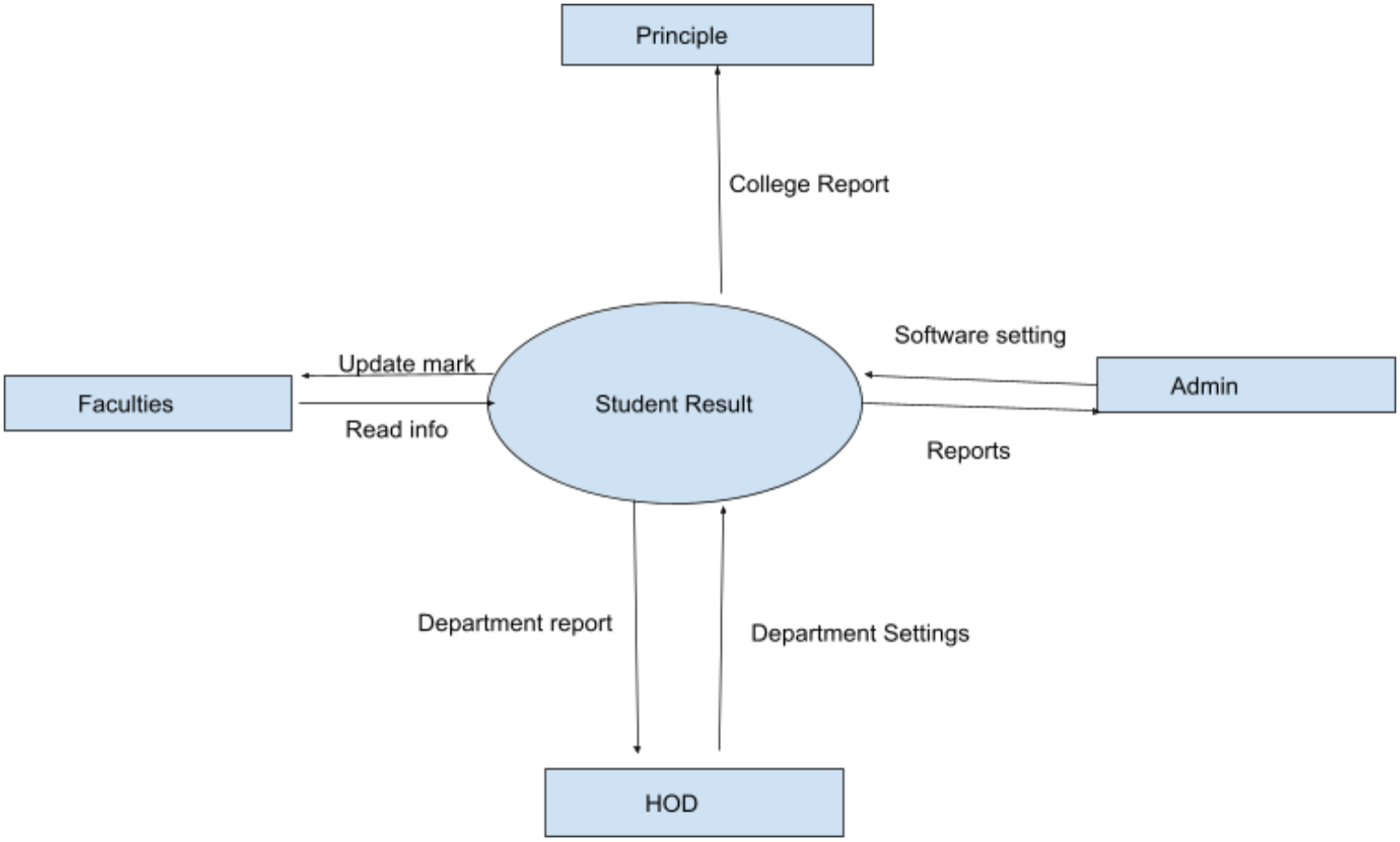
A software project implemented this model if the project is broken down into small projects. Where each project has independent functioning. And after the complete model the final product came out.

Question 2:

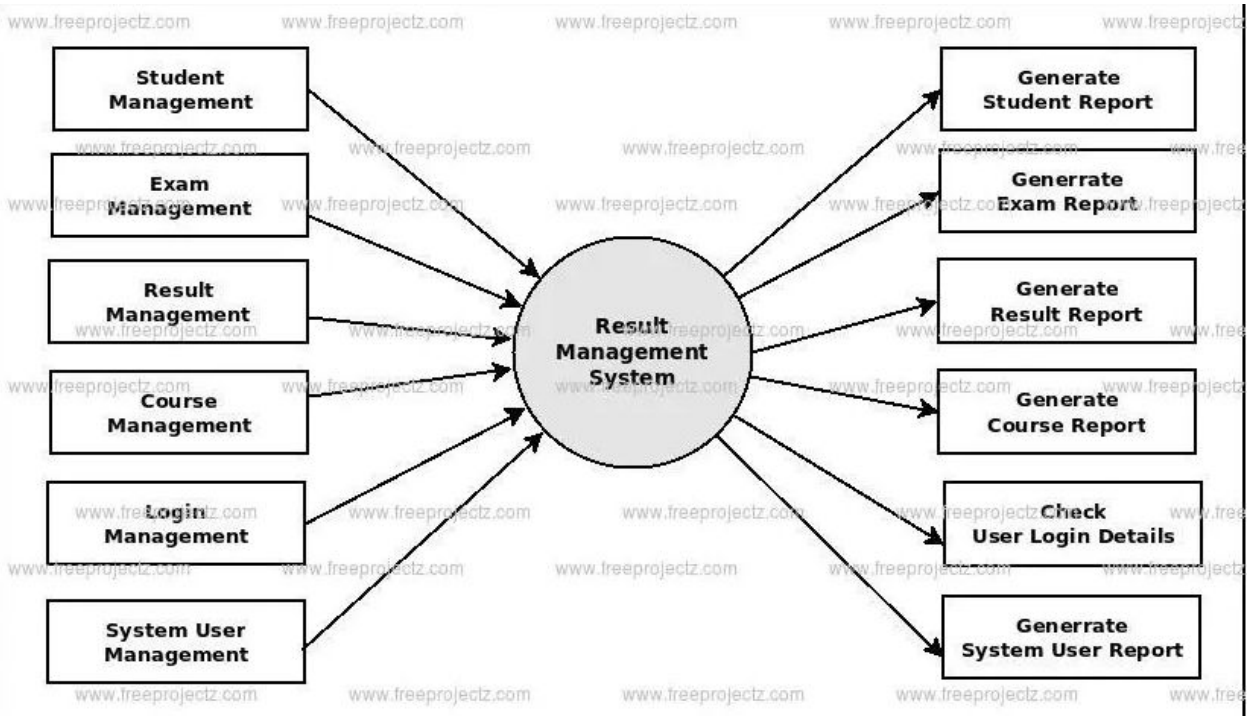
Let assume that there are 4 entities in level - 0 model

One Admin which produce and set the software for the user it set all the authentication.
Faculties can update student marks and can see info of every student related to their department.
HOD can see whole department score and can set setting for other people

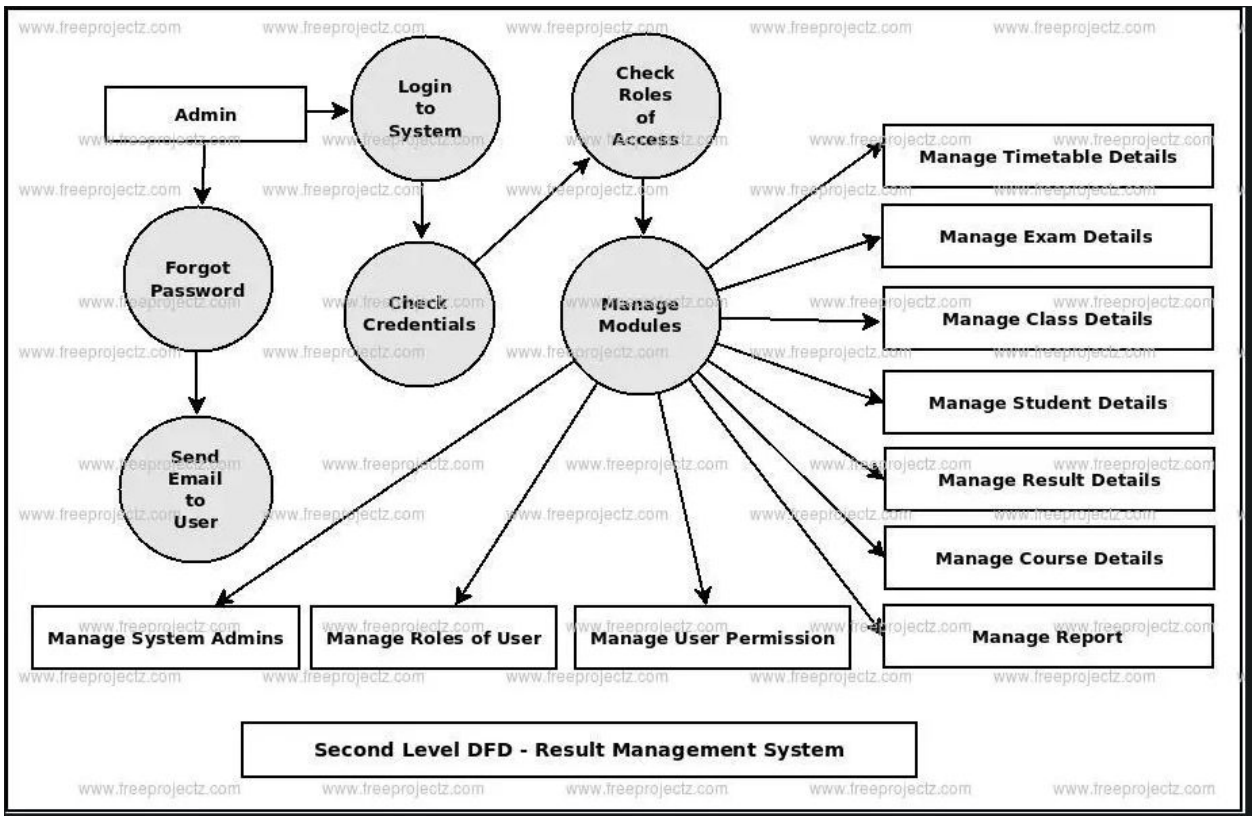
Principle can see whole college reports.


Principle
College Report
Software setting
Update mark
Faculties
Student Result
Admin
Read info
Reports
Department report
Department Settings
HOD

Level -1 :


Student Management
Exam Management
Result Management
Course Management
Login Management
System User Management
Result Management System
Generate Student Report
Generrate Exam Report
Generate Result Report
Generate Course Report
Check User Login Details
Generrate System User Report

Level 2 :


Admin
Login to System
Check Roles of Access
Forgot Password
Check Credentials
Manage Modules
Send Email to User
Manage Timetable Details
Manage Exam Details
Manage Class Details
Manage Student Details
Manage Result Details
Manage Course Details
Manage Report
Manage System Admins
Manage Roles of User
Manage User Permission
Second Level DFD - Result Management System

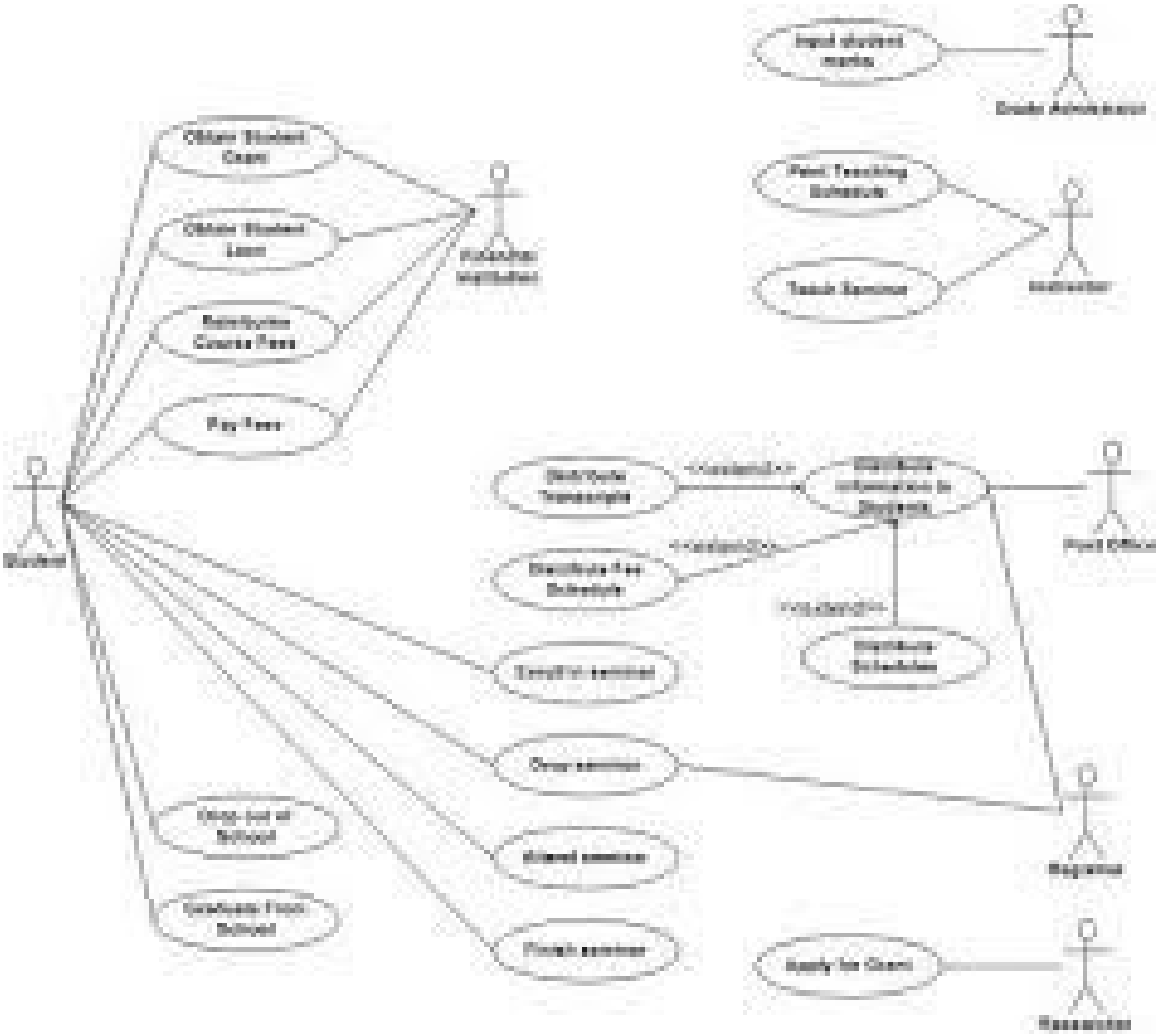
3.

The component of use case diagram are :

Actors : the users that interact with a system. An actor can be a person ,an organization, or an outside system that interacts with your application system. They must be external objects that produced or consume the data.

System : A specific sequence of actions and interactions between actors and the system.

Goals : The end result of most use cases. A diagram can describe activist and how to reach goals.



4.

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system.

Significance of Requirement Engineering

· The Requirement Engineering (RE) is the most important phase of the Software Development Life Cycle (SDLC).

· This phase is used to translate the imprecise, incomplete needs and wishes of the potential users of software into complete, precise and formal specifications.

· The specifications act as the contract between the software users and the developers. Therefore the importance of Requirement Engineering is enormous to develop effective software and in reducing software errors at the early stage of the development of software.

· Since Requirement Engineering (RE) has great role in different stages of the SDLC, its consideration in software development is crucial. There exist a number of approaches for requirement engineering.

Use of Requirement Engineering

· The purpose of requirements engineering methodologies is to make the problem that is being stated clear and complete, and to ensure that the solution is correct, reasonable, and effective.

· This chapter summarizes available representative requirements engineering methodologies, mainly focusing on the principles. · Requirements engineering approaches are processes that develop real-world problems into digital world solutions. · Each approach has its specialized thinking about the real-world problem and follows a unique process to build the system specification as the solution.

Problem in the formulation of requirements?

· Out of date – They haven’t been updated since new needs have been identified or new functionality has been included.

· Too technical – Requirements are not in the language of the end users or stakeholders, thus errors and misunderstandings occur.

· Infeasible – They mandate functionality that cannot be implemented.

· Not prioritized – Nice-to-have functionality is mixed in with must-have functionality.

· Irrelevant – They provide for features that are not needed in the final solution.

· Untraced – They are disorganized, making it difficult to determine how requirements are related to each other .

· Not built with the stakeholder in mind – Requirements are written for use by the developer or engineer, without clear understanding of what the end user needs.

· Unverifiable and not validated – They do not include verification approaches to determine whether the requirement is properly implemented in the solution.