Q1) Software Crisis & process Evaluation - Govt e-service project failure

Scenario: State govt e-Governance project for digitizing records failed due to scope creep, poor UI, & integration issues.

1) causes of Software Crisis

- Scope Creep: Requirements kept changing after development. s' started → no proper requirement management.
- Poor User Interface Design: End users ~~(govt clerks)~~ (govt clerks/citizens) found it difficult → Usability not tested.
- Integration Issues: Failed to integrate with legacy systems → lack of planning for compability & risk analysis.

2) Waterfall vs spiral Model.

- Waterfall: Linear, rigid, not good with evolving requirements, Late risk detection, Poor for integration issues.
- Spiral: Iterative, include risk analysis, stakeholder involvement at each cycle, supports involving requirement of integration testing

★ Spiral Model is better here because it allows risk management, gradual integration with legacy system, & handling requirement changes.

3. Process Improvement / Tools :

- Use Agile principles → frequents iteration, ~~with legacy system~~ stakeholder, feedback
- & Apply PSP/TSP → personal & team based process improvement, better quality assurance.
- Strong Requirement Management Tools → Control scope creep
- Early prototyping & Usability Testing → avoid UI failure.

Q2 Life Cycle Model selection Hospital Information System (HIS)

1. prototype ~~Model~~ Vs Evolutionary Model
   - prototype Model Builds quick prototypes for Users feedback, then refined into final system Good for understanding nuclear Requirements.
   - Revolutionary Model Develops system in increments, each increment adds features, ensure compliance & phased delivery.

2. Better Model

Evolutionary Model is better because :

- Handles strict healthcare regulations / compliance tested phase wise)
- Allows continuous stakeholder feedback.
- Supports phased delivery (records → billing → pharmacy)

3 Requirement Elicitation & Risk Handling

- Early elicitation workshops with Doctors, admin staff, and IT dept do gather requirements.

- Risk Handling : Each increment tested for compliance & integration, reducing risk of failure.

Q3 Requirement Engineering Smart city parking App

1 Requirement Elicitation Techniques.

- Municipalities: Intensives & workshop policy, revenue, regulation).
- Drivers: surveys, questionnaries, focus grops (Usability, mobile features).
- Traffic police: Observation, field studies (real-time monitoring needs)

2 context-level DFD:

- Entities: Drivers, Muncipality, IOT Sensors, Traffic police
- System : Smart parking App
- Flows:
  - Drivers ↔ App: parking request, Availability info
  - IOT Sensors → App: Real-time parking status.
  - Municipality ↔ App: policy data, reports.
  - Traffic police ↔ App: Violations, alerts.

3 Requirements

- Functional:

a) Display available parking slots in real-time.
b) Allow booking of slots.
c) provide route guidance to nearest slots.
d) Send notation alerts to traffic police.
e) Generate reports for municipalities.

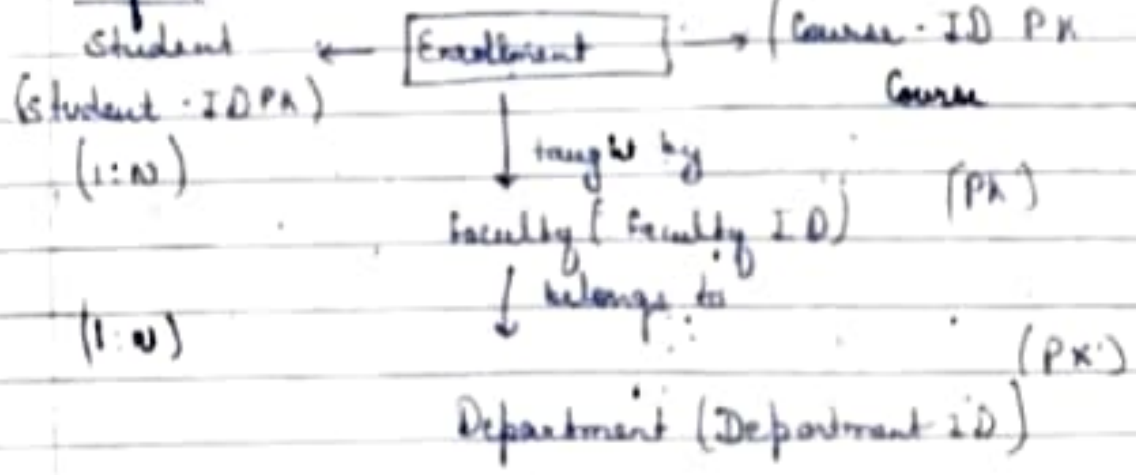
- NON Functional

a) High availability (uptime ≥99.1.)
b) Fast response time (<2 seconds)
c) Secure data transmission (incription, GDPR compliance)

Relationships:

- A student can enroll in many courses, & a course can have many students → Enrollment (M:N)
- A course is taught by one faculty (1:N)
- A Faculty belongs to one Department

Diagram



2. Data Dictionary Table

Entity	Attributes	Primary Key	Foreign Key
Student	Student · ID, Name, Email, Phone, DOB	Student ID	
Course	Course · ID, Course Name, Credits	Course ID	Faculty ID
Faculty	Faculty · ID, Name, Email	Faculty · ID	Dept · ID
Department	Dept · ID, Dept · Name, location	Dept · ID	-