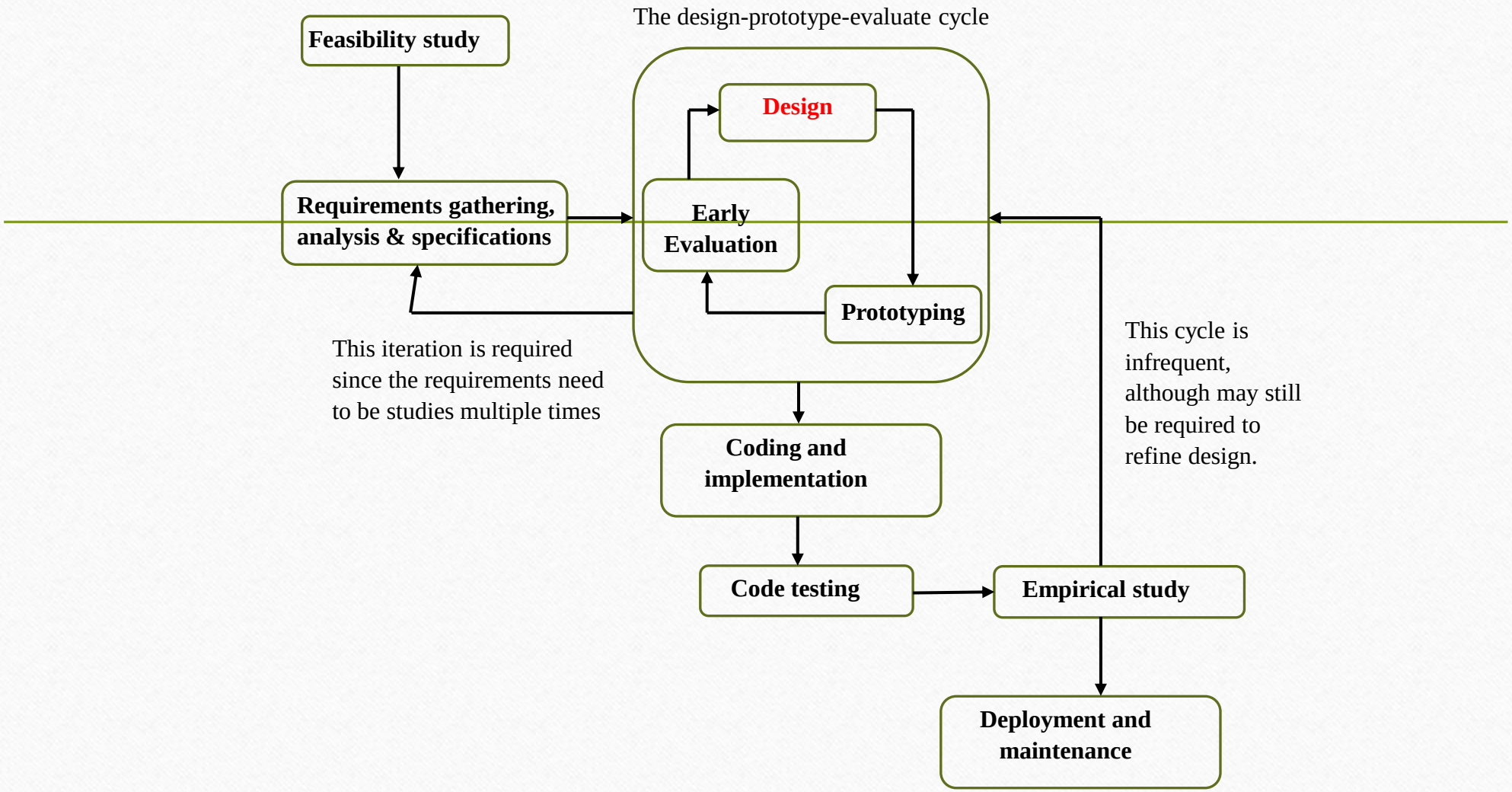
Design

Dr Samit Bhattacharya

Computer Science and Engineering

IIT Guwahati


The design-prototype-evaluate cycle
Feasibility study
Requirements gathering, analysis & specifications
Design
Early Evaluation
Prototyping
This iteration is required since the requirements need to be studies multiple times
Coding and implementation
Code testing
Empirical study
This cycle is infrequent, although may still be required to refine design.
Deployment and maintenance

Design

- Two issues
  - Where to start?
  - How to represent (design language)?

Where to Start?

- Creative thinking!
  - May be aided by intuition
  - May be aided by experience/domain knowledge

Where to Start?

- In UCD (user-centered design), we have TWO things
  - Interface design
  - Code design

Where to Start?

- Can make use of some guidelines/thumb rules/checklists/heuristics as starting point (to aid our creative thinking)

Interface Design

Interface Design Guidelines

- Mainly for GUIs
- NPTEL MOOCS course on user-centric computing for human-computer interaction, L6 (discusses only one set of guidelines: 8 golden rules; there are many more such sets)

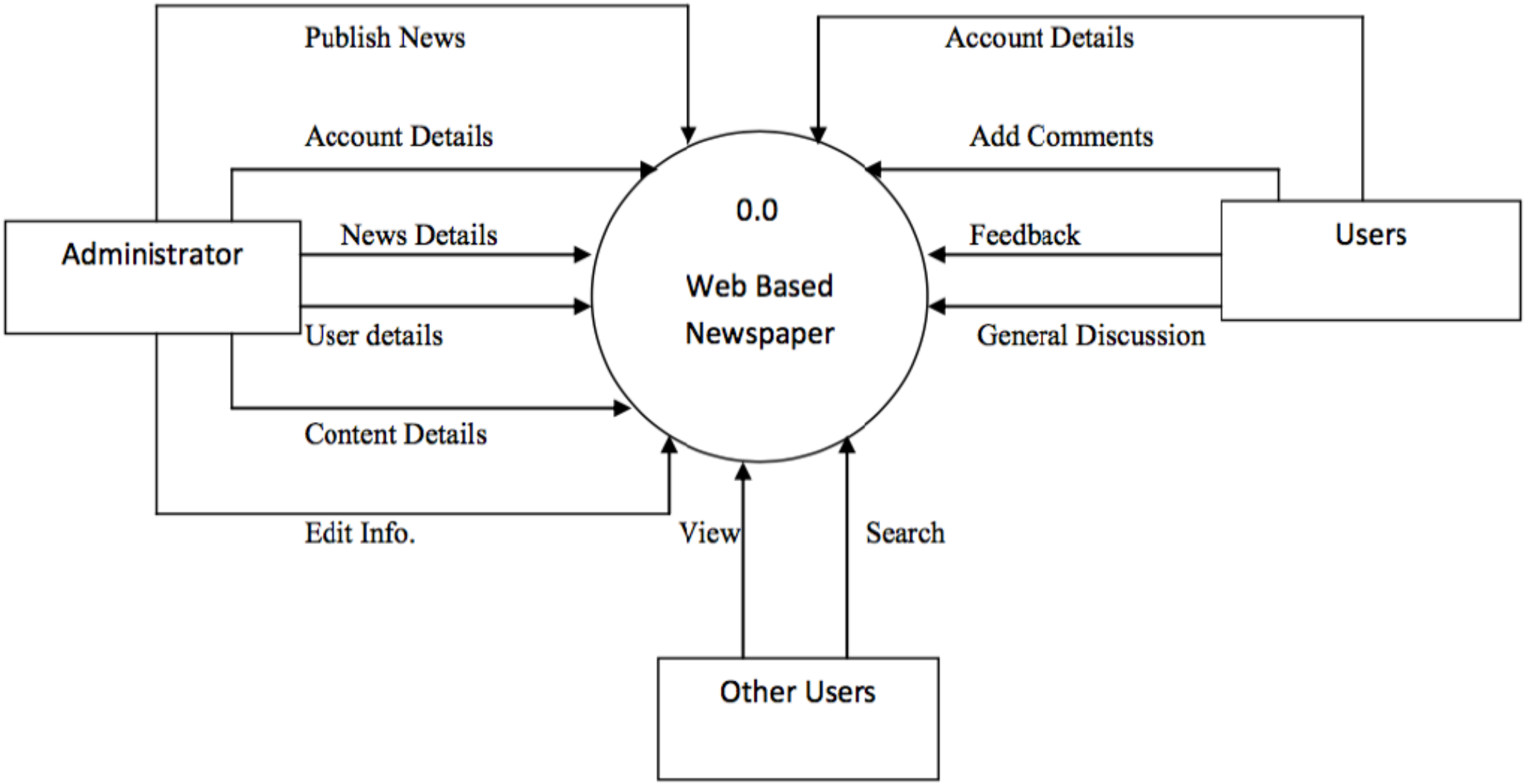
Code Design

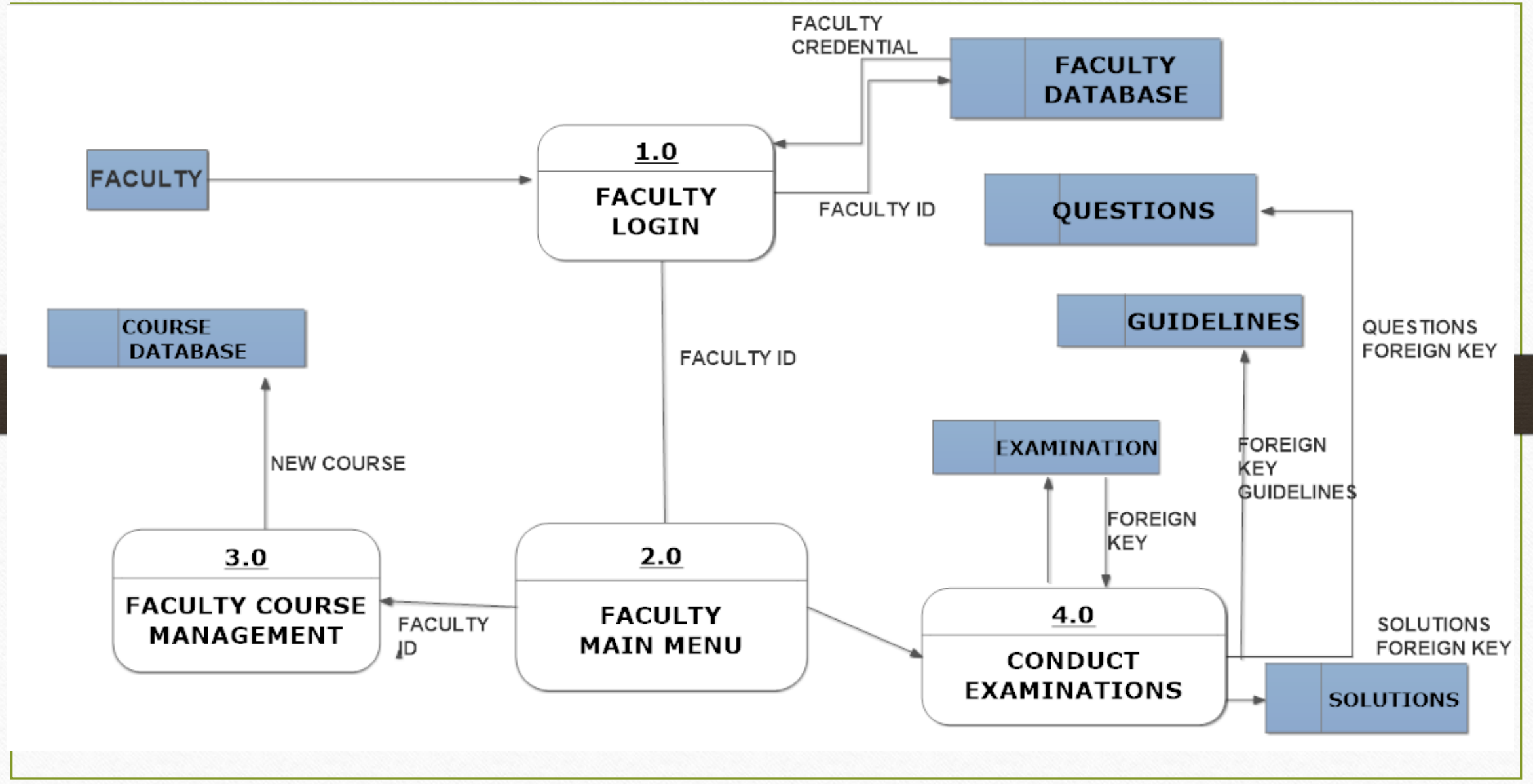
Code Design

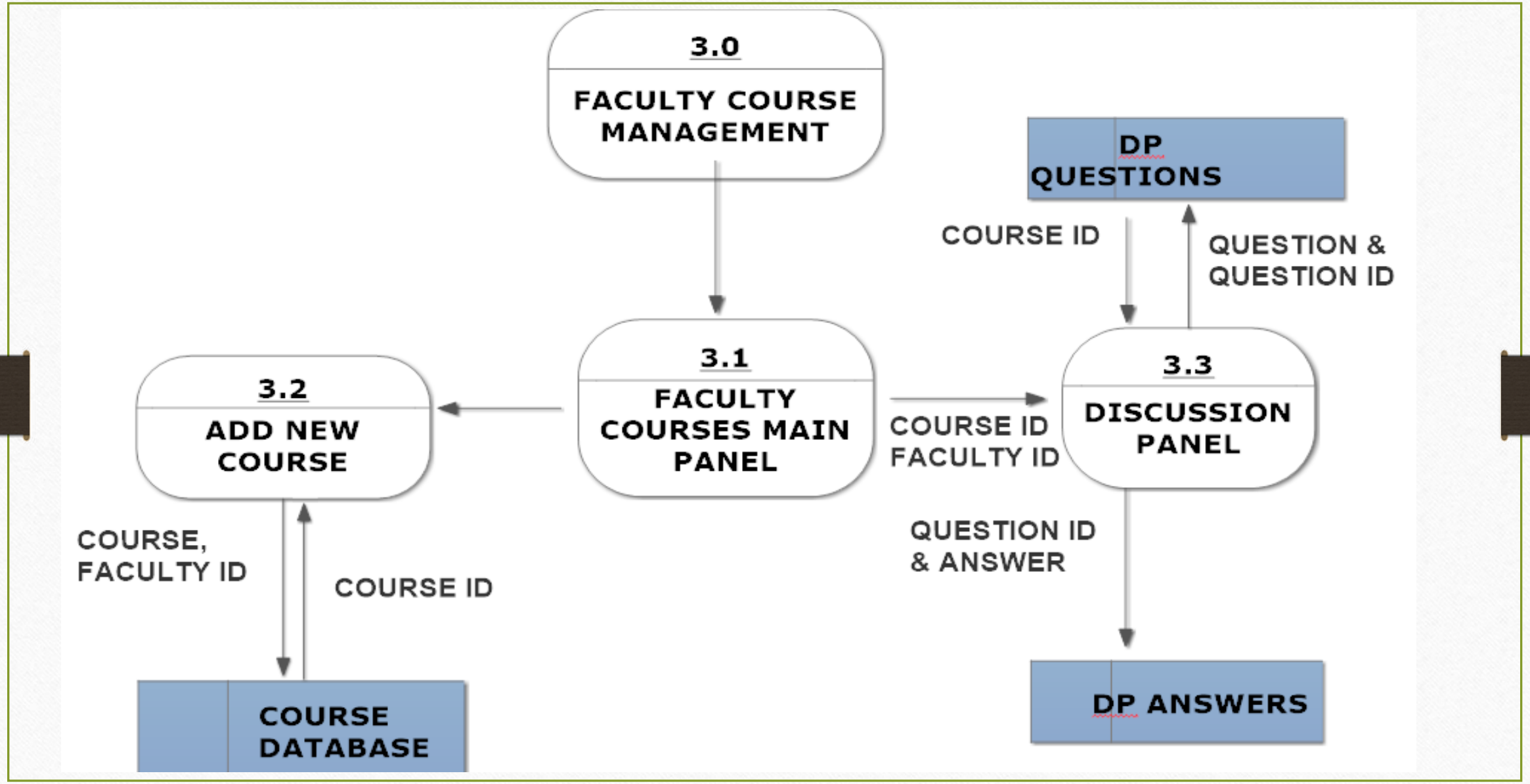
- Done based on SRS
- Two phases
  - Preliminary (high-level) design – may follow tree structures to represent modules
  - Detailed design (also called module-specification document)

High-Level Design

- Identification of modules
- Control relationships between modules
- Definition of interfaces between modules


Publish News
Account Details
Account Details
Add Comments
Administrator
News Details
User details
Content Details
Edit Info.
0.0
Web Based
Newspaper
Feedback
General Discussion
Users
View
Search
Other Users


FACULTY CREDENTIAL
FACULTY DATABASE
FACULTY
1.0
FACULTY LOGIN
FACULTY ID
QUESTIONS
GUIDELINES
QUESTIONS FOREIGN KEY
COURSE DATABASE
FACULTY ID
EXAMINATION
FOREIGN KEY GUIDELINES
NEW COURSE
FOREIGN KEY
3.0
FACULTY COURSE MANAGEMENT
FACULTY ID
2.0
FACULTY MAIN MENU
4.0
CONDUCT EXAMINATIONS
SOLUTIONS FOREIGN KEY
SOLUTIONS


3.0
FACULTY COURSE MANAGEMENT
DP QUESTIONS
COURSE ID
QUESTION & QUESTION ID
3.1
FACULTY COURSES MAIN PANEL
3.2
ADD NEW COURSE
3.3
DISCUSSION PANEL
COURSE ID
FACULTY ID
COURSE, FACULTY ID
COURSE ID
QUESTION ID & ANSWER
COURSE DATABASE
DP ANSWERS

Detailed Design

- Identification of data structures and algorithms for different modules

Characteristics of Good Design

- Coverage – should implement ALL functionalities of SRS
- Correctness - should CORRECTLY implement all functionalities of SRS
- Understandability - easily understandable (by other team members)
- Efficiency - should be efficient (in terms of resources required to implement)
- Maintainability – should be amenable to change

Cohesion and Coupling

- Good software design
  - Clean decomposition of the problem into modules
  - Neat arrangement of these modules in a hierarchy
- **Modularization depends on *cohesion & coupling***

Cohesion (of a Module)

- **Logical** – if all functions perform similar operations (e.g., error handling)
- **Temporal** – if all functions should be performed in the same time span (e.g., initialization module)
- **Procedural/functional** – if all functions are part of the same procedure (algorithm) (e.g., decoding algorithm)
- **Communication** – if all functions refer to or update same data structure (e.g., a set of functions operating on a linked list)
- **Sequential** – output from one element is input to the next element of the module (e.g., the sequence of functions get-input, validate-input, sort-input)

Coupling (between Modules)

- **Data** – if two modules communicate through a data item (e.g., passing an integer between two modules)
- **Control** – if data from one module is used to control the flow of instructions in the other module (e.g., flag setting)
- **Content** – if two modules share code (e.g., branch from one module to another)

Cohesion and Coupling

- High cohesion & low coupling → functionally independent modules

Basic Design Approaches

- Function oriented – basic abstractions are functions [use DFD to represent design]
- Object oriented – basic abstractions are objects (instantiation of class; similar objects refer to "class") [use UML to represent design]

Book

- Rajib Mall – Fundamentals of S/W Engineering
- Roger Pressman –S/W Engineering: A Practitioner's Approach
- NPTEL MOOCS course on user-centric computing for HCI, L6
- Samit Bhattacharya - Human-Computer Interaction: User-Centric Computing for Design, McGraw-Hill India (Chapter 2, Sec 2.4.4)