# Assignment 02

# SOFTWARE REENGINEERING

SE-4001

Course Instructor:

Dr. Isma Ul Hassan


**Group Members:**

| | |
|---|---|
| Zeeshan Ali | 20i-2465 |
| Ans Zeshan | 20i-0543 |
| Saad Shafiq | 20i-1793 |
| Hammad Aslam | 20i-1777 |
| Dayyan Shahzad | 20i-2393 |


Due Date:

Oct 29, 2023

# Doctor Appointment Booking System

## Re-engineering Strategies and Approaches (Part-1):

## User Experience Enhancement:

- **Approach:** By correcting the styling components of the Interface, to meet the standards that the system gives a feel for the real-time usage.
- **Rationale:** Improving the user interface enhances user engagement and satisfaction, making it more accessible and user-friendly.
- **Risks:**

  **Development Complexity**

  **Compatibility Issues**:

  **User Adaptation**:

## Security Improvement:

- **Approach:** Implement security measures, such as input validation, encryption, and authentication, to protect against SQL injection and other security vulnerabilities.
- **Rationale**: Security is essential to protect sensitive data and prevent unauthorized access or data breaches.
- **Risks:**

  **Development Effort**:

  Implementing security measures can be time-consuming and may require code changes across the system.

  **User Experience:**

  Depending on the security measures, user experience may be impacted (e.g., additional authentication steps).

## Exception Handling:

- **Approach:** Implement robust exception handling to validate input and provide meaningful error messages.
- **Rationale**: Exception handling ensures predictable behaviour and better user feedback, improving the overall quality of the system.
- **Risks:**

  **Code Complexity**:

  Extensive exception handling can make the code more complex and harder to maintain.

  **Error Propagation**:

  Poorly implemented exception handling can lead to unexpected error propagation.

## Comments for code:

- **Approach:** Add comments throughout the codebase to document functionality, making it more understandable for developers.
- **Rationale:** Comments improve code readability and help developers understand the code's purpose and logic.
- **Risks:**

  **Maintaining Comments**:

  Ensuring that comments stay accurate and up to date can be challenging, leading to potential confusion.

## Dependency Mapping:

- **Approach:** Review and update dependencies, ensuring compatibility with the latest versions using dependency management tools.
- **Rationale:** Keeping dependencies up to date ensures system stability and security.
- **Risks:**

  **Compatibility Issues**:

  Updating dependencies may introduce compatibility issues with the existing code.

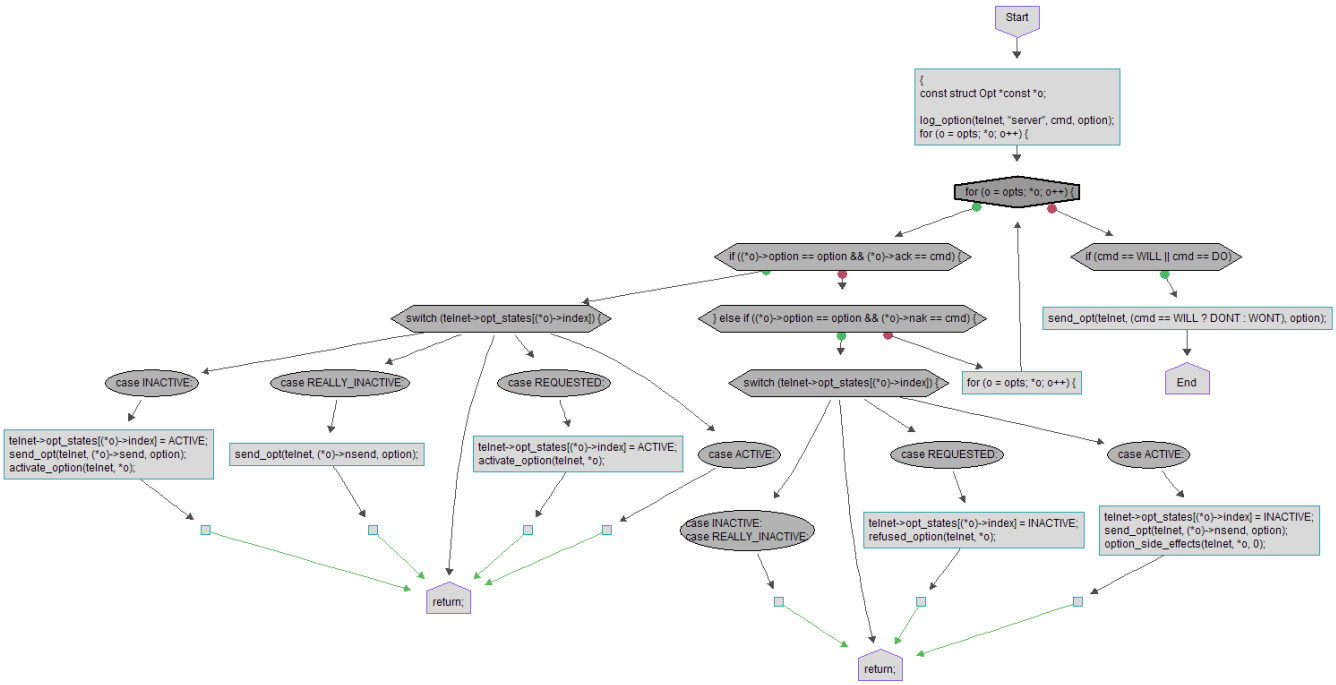
  **Testing Overhead:**

  Extensive testing is required to ensure that updated dependencies work as expected.

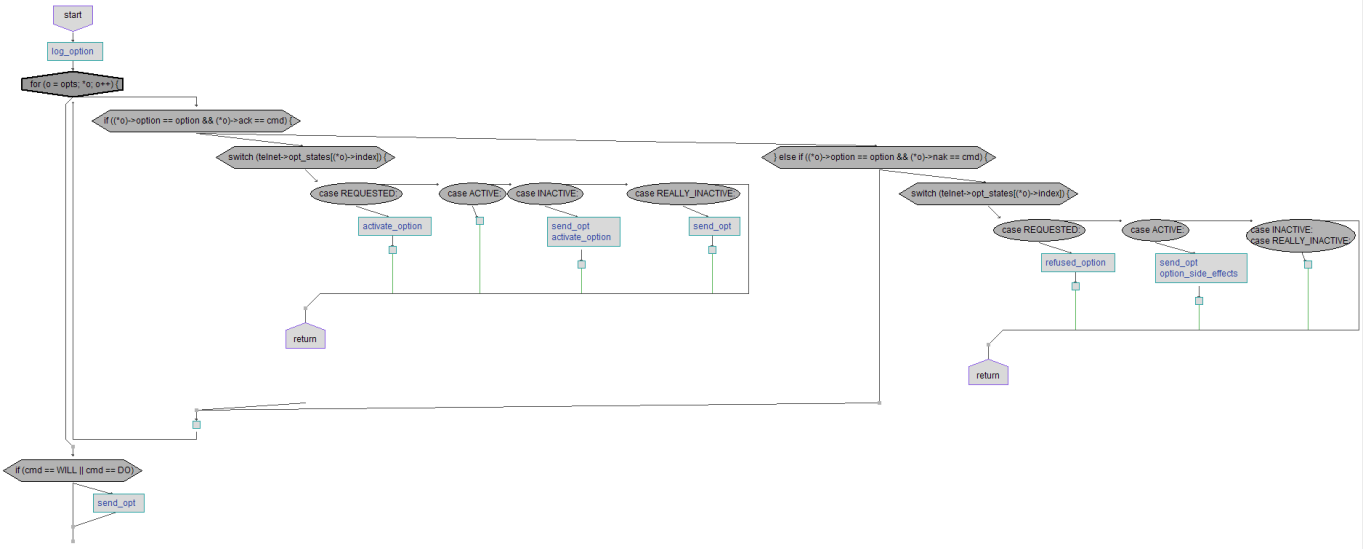
## ------- Reverse Engineering (Part-2) -------

## Source Code Analysis (Part-2a):

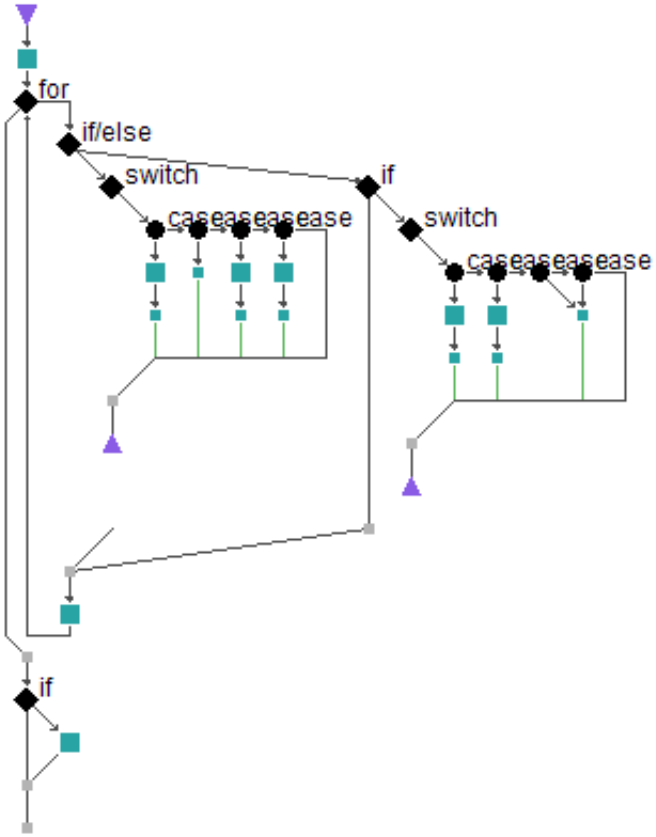
### 1- Control Flow graph:



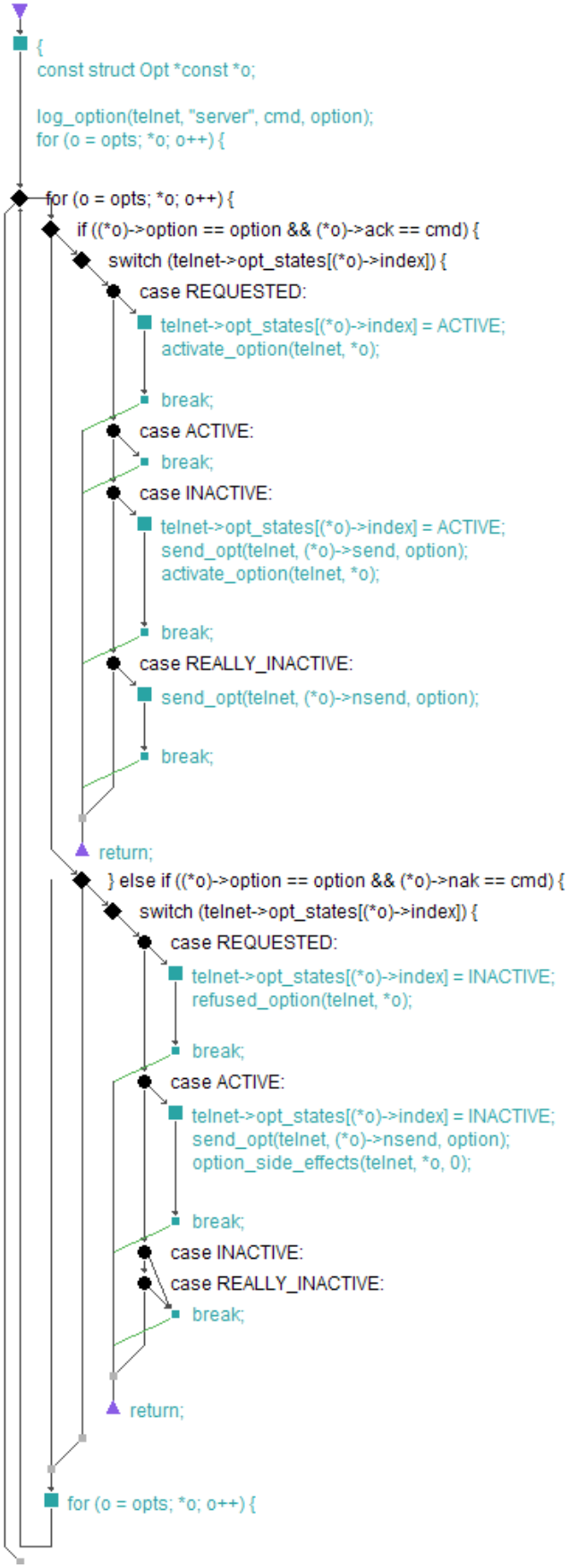
## 2- Control Dependency Graph:




## 3- Program Dependency Graph:

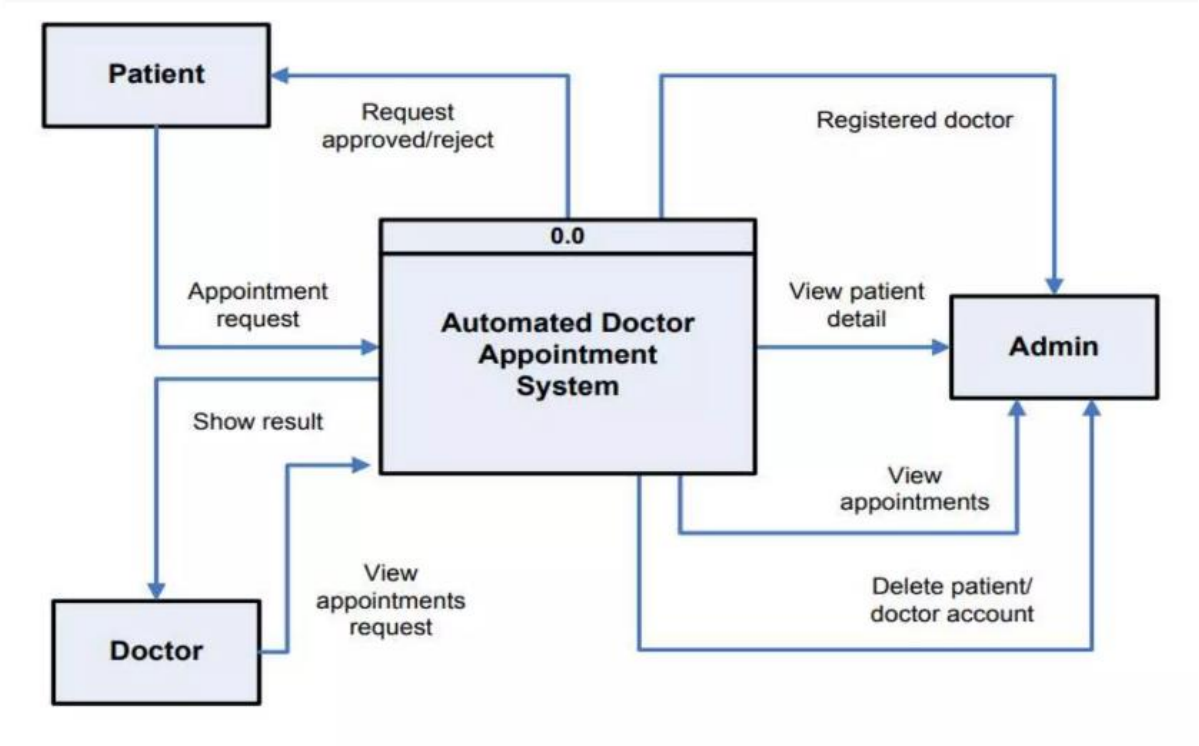


## 4- Data Dependency Graph:



## Architecture recovery and extract Design Information (Part-2b):

## Architecture:




## Design Information

**UI Components:**

**Text Field:** Used for inputting information like name, email, etc.

**Button:** Used for triggering actions like registration, login, etc.

**Date Picker:** Used for selecting date for the appointment.

**Table:** Used for displaying information in a tabular format.

**List:** Used for displaying a list of options.

**Navigation Bar:** Used for providing access to different pages within the application.

**Navigation Flow:**

1- Registration/Login Page -> Home Page -> View Patient Details Page -> Appointment Request Page -> Appointment Result Page.
2- Registration/Login Page -> Home Page -> View Doctors Page -> Book Appointment Page -> Appointment Result Page.
3- Registration/Login Page -> Home Page -> Admin Dashboard -> Delete Patient/Doctor Account Page.

**Responsive Design:** Ensuring that the application is accessible and functional on different devices (desktop, tablet, mobile) and screen sizes.

**User Privacy and Security:** Implementing appropriate security measures to protect user data and prevent unauthorized access.

**Scalability:** Ensuring that the application can handle increased user load and data storage efficiently.

**Accessibility:** Designing the application to be usable by individuals with disabilities, including the visually impaired and the hearing impaired.

**Testing and Quality Assurance:** Thoroughly testing the application to identify and fix any bugs, vulnerabilities, or issues before release.

**Performance:** Optimizing the application for faster load times, smoother interactions, and improved performance on various devices and networks.

## Database Refactoring and restructuring:

### For Database Restructuring:

We need features that are new to our system to fit in or there's a change in the schema for that functionality. As far as improvements are concerned, we have not committed to implementing any new feature. So, it hasn't made any new changes to our database schema so far to restructure it.

### For Database Refactoring:

For refactoring, there's two components.

1- Application Error.
2- Performance Issues.

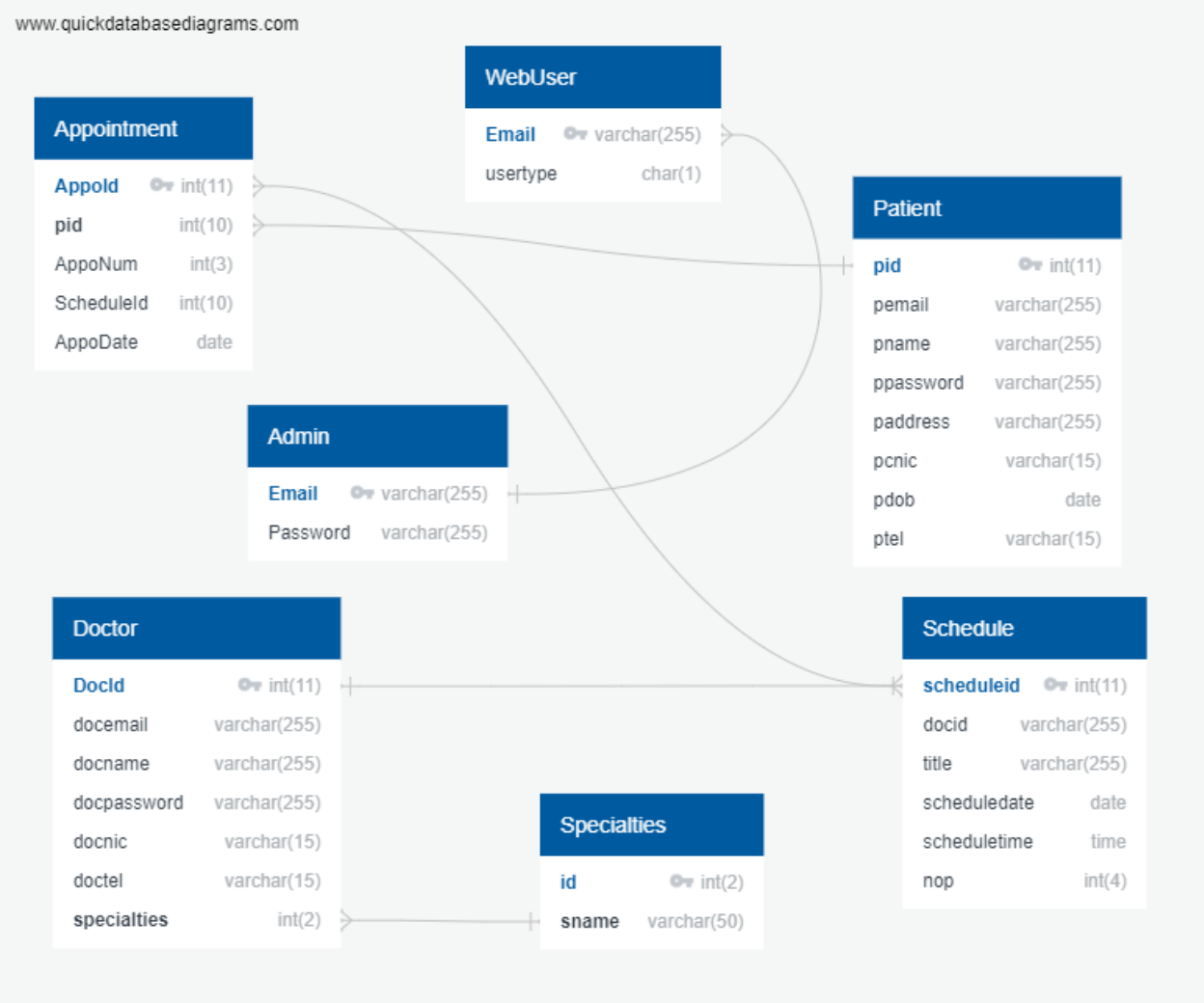
Our system gives an application error for showing the Bookings to the patient and has performance issue of retrieving the information using the SQL quires.

### Solution:

We are preventing our system from SQL injection by using the prepared statements and resolving the application error.

```
//$sql1="insert into doctor(docemail,docname,docpassword,docnic,doctel,specialties) values('$email','$name','$password','$nic','$tele',$spe
$sql1="update doctor set docemail='$email',docname='$name',docpassword='$password',docnic='$nic',doctel='$tele',specialties=$spec where doc
$database->query($sql1);

$sql1="update webuser set email='$email' where email='$oldemail' ;";
$database->query($sql1);
```




# Software Quality Assessment

### -------- Complexity Metrics --------

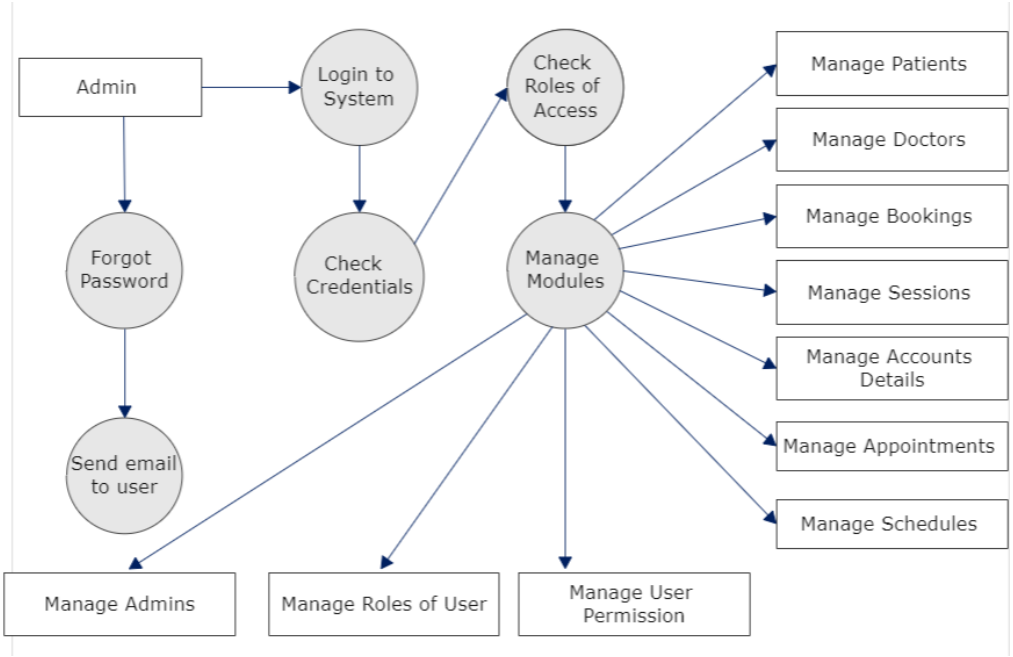
## Qualitative Code Quality Metrics

As for coding style, our code is well structed according to the module wise, and quires are also written in the simplest way to extract information. Our code demonstrates.

- **Readability and Code Formatting**
- **Clear**
- **Efficiency**

Our System is not basically an OOP Oriented Project, so there will be no such operations that are applicable in classes such as inheritance etc.

We have evaluated the Function Point Analysis of our program to check whether it is delivering all the functionality of the system or not.




The data flow diagram is evaluated to determine the key measures required for computation of the function point metric and the all metric have average weight.
number of user inputs 23 * 4 = 92
number of user outputs 20 * 5 = 100
number of user inquiries 10 * 4 = 40
number of files 3 * 7 = 21
number of external interfaces 1 * 10 = 10

UFC = 263

VAF = 28*0.01 + 0.65
VAF = 0.93

FPC = 263 * 0.93
FPC = 244.96

The software project has been assessed to have a Function Point Count (FPC) of approximately 244.59. This metric reflects the measured functionality of the system, considering factors such as the number of user inputs, outputs, inquiries, files, and external interfaces. The FPC provides a quantitative estimate of the overall size and complexity of the software.

## -------- Maintainability Metrics --------

Maintainability metrics are used to assess the ease with which a software system can be maintained and evolved over time.

- **Weighted Methods per Class (WMC):** WMC is a metric that measures the complexity of a class by counting the weighted sum of its methods. Each method is assigned a weight, typically based on its complexity or the effort required for testing. Higher WMC values suggest higher potential maintenance effort.
- **Response for a Class (RFC):** RFC measures the number of methods in a class that can potentially be executed in response to a message. It indicates the potential ripple effect when a method is changed. Higher RFC values may imply greater interconnectedness and potential impact during modifications.
- **Code Duplication:** Code Duplication measures the extent to which identical or similar code segments appear in different parts of a software system. High code duplication can lead to maintenance challenges, as changes may need to be replicated in multiple locations.
- **Coupling and Cohesion Metrics:** Coupling measures the degree of interdependence between software modules, with low coupling indicating better separation of concerns. Cohesion measures how closely the elements within a module are related. High cohesion and low coupling are generally desired for maintainable and modular code.
- **Triple Effect:** The Triple Effect refers to the ripple effect that changes in software can have on three aspects: the code itself, the dependent modules, and the documentation. It emphasizes the need to consider the broader impact of modifications.
- **Fan-in and Fan-out:** Fan-in is the count of functions or modules that call a particular function or module. Fan-out is the count of functions or modules that a particular function or module calls. Understanding fan-in and fan-out helps assess the complexity and dependencies within a software system.
- **Unit Testing:** Unit Testing is the practice of testing individual units or components of a software application in isolation. It ensures that each unit functions as intended. Effective unit testing is crucial for identifying and fixing bugs early in the development process, contributing to overall software reliability and maintainability.