12) Are there pairs of Software Quality Factors that are not independent in your system? Give an example.

Yes, there are pairs of Software Quality Factors in the system that are not entirely independent. For example:

Usability and Security

Dependency: The system's usability is influenced by its security measures. For instance, implementing role-based access control and secure login mechanisms ensures that only authorized users can access specific features. However, overly strict security measures could negatively impact usability by making the system harder to use for end-users like alumni or students.

Example in the System:

The User.php file implements secure login and session management (session\_start() and password hashing).

At the same time, the system ensures usability by allowing users to change their username and password (setUsername and setPassword methods). Balancing these two factors is critical to avoid compromising either.

Performance and Scalability

Dependency: Performance can be affected by scalability. For example, as the number of users grows, the system's database queries and event handling must remain efficient to maintain performance.

Another example in the System:

The FacultyStaff.php file includes methods for scheduling events (scheduleEvent), which interact with the database. If the number of events or participants grows significantly, the system's performance could degrade unless the database and queries are optimized for scalability.

These dependencies highlight the need to balance quality factors during design and implementation to ensure the system meets its functional and non-functional requirements effectively.

13) Calculate the LOC and CCM (Cyclomatic Complexity Metric) for the main functions in your system.

| **Function** | **LOC** | **CCM** |
| --- | --- | --- |
| Alumni::serveAsMentor() | 10 | 2 |
| Student::selectMentor() | 7 | 1 |
| FacultyStaff::scheduleEvent() | 20 | 3 |
| Event::addEvent() | 15 | 1 |
| Newsletter::publish() | 5 | 1 |
| Donation::donate() | 15 | 2 |
| Admin::createUser() | 25 | 5 |

14) For the classes in your system, calculate all the following OO Complexity Metrics (mention exactly the equation you’ve used):  
***Metric Definitions and Equations:***

| **Metric** | **Equation / Notes** |
| --- | --- |
| **WMC** (Weighted Methods per Class) | WMC = Σ (Cyclomatic Complexity per method) or count of methods (if equally weighted |
| **DIT** (Depth of Inheritance Tree) | DIT = Number of ancestor classes |
| **NOC** (Number of Children) | NOC = Number of Immediate child classes |
| **CBO** (Coupling Between Objects) | CBO = Unique external classes referenced |
| **RFC** (Response for Class) | RFC = Local Methods + External Method Calls |
| **LCOM** (Lack of Cohesion of Methods) | LCOM = 1 - (Σ(MA) / (M × A)), set to 0 if < 0 |

* **M** = Number of methods in class
* **A** = Number of attributes
* **MA** = Number of attributes used per method

***Class Metrics Table:***

| **Class** | **WMC** | **DIT** | **NOC** | **CBO** | **RFC** | **LCOM** |
| --- | --- | --- | --- | --- | --- | --- |
| User | 9 | 1 | 4 | 3 | 14 | 0.33 |
| Admin | 6 | 2 | 0 | 2 | 9 | 0.40 |
| Alumni | 12 | 2 | 0 | 5 | 18 | 0.42 |
| FacultyStaff | 8 | 2 | 0 | 4 | 14 | 0.38 |
| Student | 7 | 2 | 0 | 4 | 12 | 0.36 |
| Event | 9 | 1 | 0 | 3 | 12 | 0.30 |
| Mentorship | 10 | 1 | 0 | 3 | 14 | 0.40 |
| Newsletter | 8 | 2 | 0 | 4 | 13 | 0.35 |
| Observer | 1 | 1 | 1 | 0 | 1 | 0.00 |
| Subject | 7 | 1 | 2 | 5 | 12 | 0.50 |

14) Considering White-Box Testing, generate a Unit-Testing Test Report for at least 6 main functions in your system.

### 1. Alumni::serveAsMentor() Paths:

1. Path 1: Alumni is not verified → Exception is thrown.
2. Path 2: Alumni is verified → Mentor status is updated, and a new mentorship is created.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | verified = false | Exception: "You must be verified to serve as a mentor." |
| TC2 | verified = true | Mentor status updated, mentorship created successfully. |

### 2. Student::selectMentor(mentorId)

Paths:

1. Path 1: Database query executes successfully → Mentor is assigned.
2. Path 2: Database query fails → Exception is thrown.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | mentorId = valid ID | Mentor assigned successfully. |
| TC2 | mentorId = invalid ID | Exception: Database error. |

### 3. FacultyStaff::scheduleEvent(name, description, date)

Paths:

1. Path 1: Event with the same name already exists → Exception is thrown.
2. Path 2: Event date is in the past → Exception is thrown.
3. Path 3: Event is successfully scheduled.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | name = existing name, date = future date | Exception: "Event with this name already exists." |
| TC2 | name = new name, date = past date | Exception: "Event date cannot be in the past." |
| TC3 | name = new name, date = future date | Event scheduled successfully. |

### 4. Donation::donate()

Paths:

1. Path 1: Donation is successfully recorded in the database.
2. Path 2: Database query fails → Exception is thrown.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | donorId = valid ID, amount = 100 | "Donation of 100 for [cause] has been made successfully." |
| TC2 | donorId = invalid ID, amount = 100 | Exception: Database error. |

### 5. Event::getEventById(eventId)

Paths:

1. Path 1: Event ID exists → Event details are returned.
2. Path 2: Event ID does not exist → Exception is thrown.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | eventId = valid | Event details are returned. |
| TC2 | eventId = invalid | Exception: "Event not found." |

### 6.Admin::createUser(username, password, role)

Paths:

1. Path 1: Input validation fails → Exception is thrown.
2. Path 2: Username already exists → Exception is thrown.
3. Path 3: User is successfully created.

Test Cases:

| **Test Case ID** | **Input** | **Expected Output** |
| --- | --- | --- |
| TC1 | username = "", password = "pass", role = "Admin" | Exception: "Username, password, and role cannot be empty." |
| TC2 | username = "existingUser", password = "pass", role = "Admin" | Exception: "Username already exists." |
| TC3 | username = "newUser", password = "pass", role = "Admin" | User created successfully. |

16)Considering Black-Box Testing, generate a Functional System-Testing Test Report for at least 6 main functions in your system.  
  
  
1. Alumni::serveAsMentor()  
Description: Allows an alumni user to serve as a mentor if they are verified.

Boundary Conditions:

1. Alumni is not verified.
2. Alumni is verified.

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | verified = false | Exception: "You must be verified to serve as a mentor." |
| TC2 | verified = true | Mentor status updated, mentorship created successfully. |

### 2. Student::selectMentor(mentorId) Description: Allows a student to select a mentor by their ID.

Boundary Conditions:

1. Mentor ID is valid.
2. Mentor ID is invalid (does not exist in the database).

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | mentorId = valid ID | Mentor assigned successfully. |
| TC2 | mentorId = invalid ID | Exception: "Mentor not found." |

### 3. FacultyStaff::scheduleEvent(name, description, date) Description: Allows faculty staff to schedule an event.

Boundary Conditions:

1. Event name already exists.
2. Event date is in the past.
3. Event date is in the future.

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | name = existing name, date = future date | Exception: "Event with this name already exists." |
| TC2 | name = new name, date = past date | Exception: "Event date cannot be in the past." |
| TC3 | name = new name, date = future date | Event scheduled successfully. |

### 4. Donation::donate(amount, cause)

Description: Allows alumni to make a donation for a specific cause.

Boundary Conditions:

1. Donation amount is below the minimum allowed (ex, $1).
2. Donation amount is exactly the minimum allowed.
3. Donation amount is above the minimum allowed.

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | amount = 0, cause = "Education" | Exception: "Donation amount must be at least $1." |
| TC2 | amount = 1, cause = "Education" | "Donation of $1 for Education has been made successfully." |
| TC3 | amount = 100, cause = "Education" | "Donation of $100 for Education has been made successfully." |

### 5. Event::getEventById(eventId)

File: src/Event.php  
Description: Retrieves event details by its ID.

Boundary Conditions:

1. Event ID exists.
2. Event ID does not exist.

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | eventId = valid | Event details are returned. |
| TC2 | eventId = invalid | Exception: "Event not found." |

### 

### 

### 6. Admin::createUser(username, password, role)

Description: Allows an admin to create a new user.

Boundary Conditions:

1. Username is empty.
2. Password is empty.
3. Role is invalid.
4. Username already exists.
5. All inputs are valid.

Test Cases:

| Test Case ID | Input | Expected Output |
| --- | --- | --- |
| TC1 | username = "", password = "pass", role = "Admin" | Exception: "Username, password, and role cannot be empty." |
| TC2 | username = "newUser", password = "", role = "Admin" | Exception: "Username, password, and role cannot be empty." |
| TC3 | username = "newUser", password = "pass", role = "InvalidRole" | Exception: "Invalid role specified." |
| TC4 | username = "existingUser", password = "pass", role = "Admin" | Exception: "Username already exists." |
| TC5 | username = "newUser", password = "pass", role = "Admin" | User created successfully. |