UNIVERSITY OF CENTRAL ARKANSAS
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Requirements to Design to Testing: Class Activity - From Requirements to Implementation

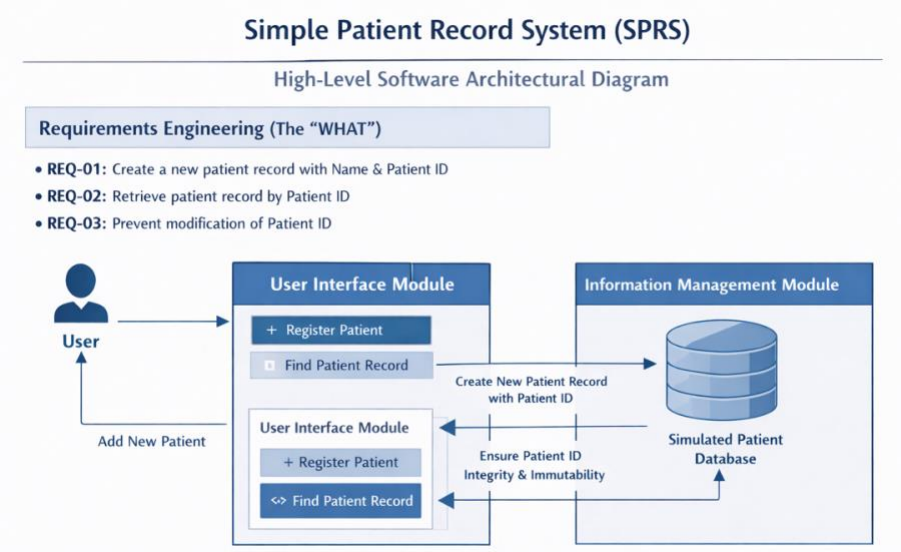
**Scenario: Simple Patient Record System (SPRS) -** This activity is designed to help students practice **Design Realization**, which is the process of translating software design into high-quality, functional code while maintaining strict **traceability** back to requirements.

**Phase 1: Requirements Engineering (The "WHAT")**
The system must manage basic patient data. Students are given the following specific requirements:

1. **REQ-01:** The system shall allow the creation of a new patient record with a name and a unique Patient ID.
2. **REQ-02:** The system shall retrieve a patient's record using their unique Patient ID.
3. **REQ-03:** The system shall ensure data integrity by preventing the modification of a Patient ID once it has been assigned.

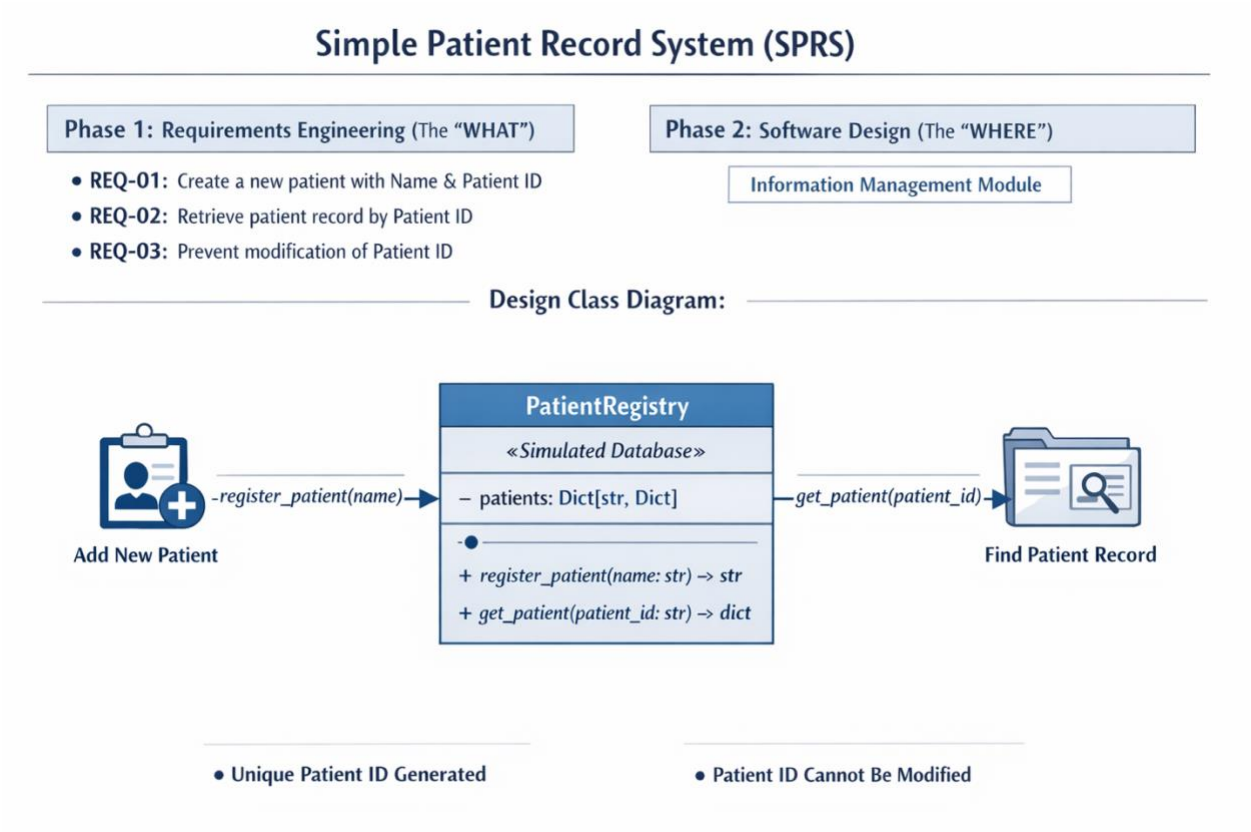
**Phase 2: Software Design (The "WHERE")**
To realize these requirements, the high-level architecture involves an **Information Management Module**. As learned in the lecture, architectural components must be mapped to source code modules.

**Design a High-Level Diagram:**



**Design Class Diagram:**

- **Class:** PatientRegistry
- **Responsibilities:**
  - Maintain a collection of patients (Simulated Database).
  - Provide a public interface for adding and finding patients while practicing **encapsulation** and **information hiding**.
- **Methods:**
  - register_patient(name: str) -> str: Generates a unique ID and stores the patient.
  - get_patient(patient_id: str) -> dict: Returns patient data or an error if not found.



**Phase 3: Student Implementation Task (The "HOW")**
Students should work in pairs to implement the design while following **professional coding standards**.

**Your Tasks:**

1. **Create a module/file:** patient_registry.py.
2. **Implement the class:** PatientRegistry.
3. **Implement the logic:**
   - Use a dictionary to simulate a database.
   - **REQ-01 & REQ-03:** In register_patient, generate a unique ID (e.g., "P-101"). Ensure the ID cannot be changed later (simulated by not providing a "setter" for the ID).
   - **REQ-02:** In get_patient, handle cases where the ID does not exist using proper **error-handling guidelines**.
4. **Maintain Traceability:** You **must** include comments in your code linking specific methods to the Requirement IDs (e.g., # REQ-01).
5. **Write Unit Tests:** Create a test file test_patient.py to verify:
   - A patient can be successfully registered and retrieved.
   - Searching for a non-existent ID returns an appropriate error.

**Phase 4: Implementation Principles & Quality**
Students must ensure their implementation exhibits the **characteristics of clean code**:

- **Readability:** Use descriptive naming conventions (e.g., PEP8 for Python).
- **Single Responsibility:** Ensure the PatientRegistry class only handles patient storage and retrieval.
- **Self-Documenting Code:** While good code should be clear, include **API documentation** or brief comments to explain complex logic.

***Run the program: https://www.online-python.com/***

**Initial Solution (patient_registry.py):**

```python
# patient_registry.py

class PatientRegistry:
    def __init__(self):
        # Simulated database: patient_id -> patient data
        self._patients = {}
        self._next_id = 101  # starting ID number

    # REQ-01 & REQ-03
    def register_patient(self, name):
        # Create a new patient with unique ID
        if not name:
            raise ValueError("Name cannot be empty.")

        patient_id = f"P-{self._next_id}"
        self._next_id += 1

        # Store patient record (ID not modifiable later)
        self._patients[patient_id] = {
            "patient_id": patient_id,
            "name": name
        }

        return patient_id

    # REQ-02
    def get_patient(self, patient_id):
        # Retrieve patient by ID
        if patient_id not in self._patients:
            raise LookupError(f"No patient found with ID '{patient_id}'")

        return self._patients[patient_id]
```

```
# Demo for online compiler
if __name__ == "__main__":
    registry = PatientRegistry()

    pid = registry.register_patient("Alice")
    print("Registered:", pid)

    print("Retrieved:", registry.get_patient(pid))

    # Example error handling
    try:
        registry.get_patient("P-999")
    except LookupError as e:
        print("Error:", e)
```

**Test Script:**

```
# test_patient.py

import unittest
from patient_registry import PatientRegistry


class TestPatientRegistry(unittest.TestCase):

    def setUp(self):
        # Create a new registry before each test
        self.registry = PatientRegistry()

    def test_register_and_retrieve_patient(self):
        # Register patient (REQ-01)
```

```
        pid = self.registry.register_patient("Alice")

        # Retrieve patient (REQ-02)
        record = self.registry.get_patient(pid)

        # Verify data
        self.assertEqual(record["patient_id"], pid)
        self.assertEqual(record["name"], "Alice")

    def test_non_existent_id_raises_error(self):
        # Try retrieving unknown ID (REQ-02 error handling)
        with self.assertRaises(LookupError):
            self.registry.get_patient("P-999")


if __name__ == "__main__":
    unittest.main()
```

Final Runnable Solution:

```
# patient_registry.py

class PatientRegistry:
    def __init__(self):
        self._patients = {}   # simulated database: patient_id -> {"patient_id":..., "name":...}
        self._next_id = 101   # first generated ID: P-101

    # REQ-01 & REQ-03
    def register_patient(self, name: str) -> str:
        if not name or not str(name).strip():
            raise ValueError("Name cannot be empty.")

        patient_id = f"P-{self._next_id}"
        self._next_id += 1
```

```
        # Store record; no method exists to change patient_id later (REQ-03)
        self._patients[patient_id] = {"patient_id": patient_id, "name": str(name).strip()}
        return patient_id

    # REQ-02
    def get_patient(self, patient_id: str) -> dict:
        pid = str(patient_id).strip()
        if not pid:
            raise ValueError("Patient ID cannot be empty.")

        if pid not in self._patients:
            raise LookupError(f"No patient found with ID '{pid}'")

        return self._patients[pid]

    # (Update allowed: name only; ID remains immutable per REQ-03)
    def update_patient_name(self, patient_id: str, new_name: str) -> dict:
        record = self.get_patient(patient_id)  # reuses REQ-02 error handling
        if not new_name or not str(new_name).strip():
            raise ValueError("New name cannot be empty.")
        record["name"] = str(new_name).strip()
        return record

    def list_patients(self) -> list:
        return list(self._patients.values())
```

```
# test_patient.py

import unittest
from patient_registry import PatientRegistry
```

```
class TestPatientRegistry(unittest.TestCase):

    def setUp(self):
        self.registry = PatientRegistry()

    def test_register_and_retrieve_patient(self):
        # REQ-01
        pid = self.registry.register_patient("Alice")

        # REQ-02
        record = self.registry.get_patient(pid)

        self.assertEqual(record["patient_id"], pid)
        self.assertEqual(record["name"], "Alice")

    def test_non_existent_id_raises_error(self):
        # REQ-02 error handling
        with self.assertRaises(LookupError):
            self.registry.get_patient("P-999")


if __name__ == "__main__":
    unittest.main()
```

```
# main.py

import unittest
from patient_registry import PatientRegistry
from test_patient import TestPatientRegistry


def run_application():
    registry = PatientRegistry()
```

```
while True:
    print("\n=== Simple Patient Record System (SPRS) ===")
    print("1) Register new patient")
    print("2) Retrieve patient by ID")
    print("3) Update patient name (ID cannot be changed)")
    print("4) List all patients")
    print("5) Go back to main")
    print("6) Exit")

    choice = input("Choose an option (1-6): ").strip()

    if choice == "1":
        try:
            name = input("Enter patient name: ").strip()
            pid = registry.register_patient(name)
            print(f"✅ Registered. Patient ID: {pid}")
        except ValueError as e:
            print(f"❌ {e}")

    elif choice == "2":
        try:
            pid = input("Enter patient ID (e.g., P-101): ").strip()
            record = registry.get_patient(pid)
            print("✅ Record:", record)
        except (ValueError, LookupError) as e:
            print(f"❌ {e}")

    elif choice == "3":
        try:
            pid = input("Enter patient ID to update: ").strip()
            new_name = input("Enter new patient name: ").strip()
            updated = registry.update_patient_name(pid, new_name)
            print("✅ Updated:", updated)
```

```
        except (ValueError, LookupError) as e:
            print(f"❌ {e}")

    elif choice == "4":
        records = registry.list_patients()
        if not records:
            print("ℹ️ No patients registered yet.")
        else:
            print("=== All Patients ===")
            for r in records:
                print(f"- {r['patient_id']}: {r['name']}")

    elif choice == "5":
        return  # go back to main menu

    elif choice == "6":
        print("Goodbye!")
        raise SystemExit(0)

    else:
        print("❌ Invalid option. Please choose 1-6.")


def run_tests():
    print("\n=== Running Unit Tests (SPRS) ===")

    suite = unittest.TestSuite()
    suite.addTest(unittest.defaultTestLoader.loadTestsFromTestCase(TestPatientRegistry))

    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)

    if result.wasSuccessful():
        print("\n✅ TEST RESULT: PASS (All tests passed)")
```

```
    else:
        print("\n❌ TEST RESULT: FAIL")
        print(f"Failures: {len(result.failures)}")
        print(f"Errors:   {len(result.errors)}")


def main():
    while True:
        print("\n=== Main Menu ===")
        print("1) Run Application")
        print("2) Run Unit Tests")
        print("3) Exit")

        mode = input("Choose (1-3): ").strip()

        if mode == "1":
            run_application()
        elif mode == "2":
            run_tests()
        elif mode == "3":
            print("Goodbye!")
            break
        else:
            print("❌ Invalid selection. Please choose 1-3.")


if __name__ == "__main__":
    main()
```