# 

| **April 16, 2025** |
| --- |
| **Unit Testing for Medicure Clinic Management System** |
| **2/3 Implementation (Beta-Beta)** |

# 

# 

# 

# **Table of Content**

[**April 16, 2025 1**](#_kuajvu6bhoiu)

[**Unit Testing for Medicure Clinic Management System 1**](#_kuajvu6bhoiu)

[**2/3 Implementation (Beta-Beta) 1**](#_kuajvu6bhoiu)

[**Table of Content 2**](#_vbszkdvi1mla)

[**1. Case Studies 3**](#_imtobpo658s7)

[1.1 OpenMRS (Open Medical Record System) 4](#_c11l053rwfh2)

[Overview 4](#_dr4kkpyoc53f)

[Unit Testing Approach 4](#_iookbxfui6pm)

[Text file 5](#_9pay1cs6li21)

[How This Relates to Medicure 5](#_hzhwsery21gc)

[Authentication & Role-Based Access Testing 5](#_xyozy4ew4z9n)

[How This Relates to Medicure 5](#_c3mhhx7dqqjl)

[Strengths and Weaknesses of OpenMRS’ Testing Approach 6](#_ald50r7r3wui)

[What Medicure Should Implement 7](#_lknd4d9j2nmj)

[What Medicure Can Improve 7](#_cla3vtidjgua)

[1.2. ClinicSys – Comprehensive Clinic Management System 8](#_jjlms2ttofvv)

[Overview 8](#_n8e21cuwaty8)

[Key Features of ClinicSys 8](#_3ekqxy9g78he)

[Unit Testing Approach in ClinicSys 8](#_9yoc3m6n00u6)

[1. Mocking Dependencies 8](#_iq8xxugqa914)

[How This Relates to Medicure 9](#_utfqkpt0eea7)

[2. Program Logic Testing 9](#_rk36mufmgzc1)

[3. Authentication & Role-Based Access Testing 10](#_b3g5oo8b8z0x)

[How This Relates to Medicure 10](#_wv3xr0f8irnj)

[Strengths and Weaknesses of ClinicSys’ Testing Approach 10](#_maojpgbyylmu)

[What Medicure Should Implement 11](#_91j78zldbol4)

[What Medicure Can Improve 12](#_1772a8xvufk9)

[1.3. MedKey – Cloud-Based Medical Practice Management 12](#_sca02cg45xw)

[Overview 12](#_7p7v58qg3y2a)

[Key Features of MedKey 12](#_dxv5kav4a50x)

[Unit Testing Approach in MedKey 13](#_utheaje16eov)

[1. Mocking Dependencies 13](#_g5h99q4ig712)

[How This Relates to Medicure 13](#_rwwp718fom46)

[2. Authentication & Role-Based Access Testing 14](#_wvmt64v6kz9m)

[How This Relates to Medicure 14](#_i4d11ldtstxs)

[Strengths and Weaknesses of MedKey’s Testing Approach 14](#_uvymklonnawi)

[What Medicure Should Implement 15](#_nfx4synap9z8)

[What Medicure Can Improve 15](#_685l29wx87t5)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **1. Case Studies**

## 1.1 OpenMRS (Open Medical Record System)

### **Overview**

OpenMRS (Open Medical Record System) is an open-source electronic medical record (EMR) system designed to improve healthcare management in hospitals and clinics, particularly in resource-limited environments. It is widely used in government hospitals, research institutions, and non-profit healthcare organizations.

The system provides core functionalities such as:

* **Patient Registration & Records Management** – Centralized storage of patient medical history.
* **Appointment Scheduling** – Manages doctor-patient scheduling and availability.
* **Billing & Payments** – Tracks patient expenses and insurance claims.
* **Authentication & Role-Based Access** – Ensures that different user roles (doctors, nurses, administrators) have appropriate access rights.

Since **Medicure** shares similar features, OpenMRS serves as an excellent reference point for designing and implementing robust unit testing strategies.

### Unit Testing Approach

Medicure is developed using C# and relies on the MSTest framework for unit testing. The testing approach focuses on isolating and validating business logic while minimizing dependencies on external systems.

One key challenge in testing systems like Medicure is the involvement of components such as databases and authentication services. To ensure that unit tests remain focused and reliable, we avoid direct interactions with these external dependencies during testing.

Instead of connecting to a real database, we design the architecture in a way that allows controlled testing of core logic. This may include the use of stubs or simple in-memory objects to simulate the behavior of external services, ensuring that unit tests remain fast, repeatable, and independent.

By structuring our tests this way, we can achieve better test coverage and verify that core functionalities behave as expected, without relying on a full backend setup.

### Text file

OpenMRS does not directly test its database using unit tests. Instead, it uses an **in-memory database** (such as H2 Database) to simulate database transactions without affecting real data. This helps validate:

* **CRUD operations (Create, Read, Update, Delete)**
* **Data consistency across multiple operations**

##### How This Relates to Medicure

Since **Medicure** will heavily rely on patient and appointment data, it makes sense to **test database operations in an isolated in-memory environment**.  
 Using dummy text files in **MSTest** will allow us to test database queries without modifying the actual clinic database.

### Authentication & Role-Based Access Testing

OpenMRS implements a strict **role-based access control (RBAC) model**, meaning different users have different levels of access. This is tested using **unit tests and integration tests** to validate:

* **Whether a doctor can access patient records but not billing information**
* **Whether an administrator can manage users but not modify medical data**
* **Whether an unauthorized user is denied access to sensitive information**

##### How This Relates to Medicure

Since Medicure also features **user authentication and role management**, we need to create unit tests that verify:

* **Doctors can only edit patient history but not modify financial records.**
* **Admins can create and deactivate user accounts but not modify medical treatments.**

Using **MSTest’s assertion methods**, we can create test cases that validate role-based access permissions before deploying the system.

### **Strengths and Weaknesses of OpenMRS’ Testing Approach**

| **Factor** | **Strengths** | **Weaknesses** |
| --- | --- | --- |
| **Testing Framework** | Uses industry-standard tools (JUnit & Mockito) for effective unit testing. | Some advanced testing scenarios require **manual validation** due to system complexity. |
| **Mocking & Isolation** | Well-implemented mock services ensure independent unit tests that are **fast and reliable**. | Some **complex scenarios (e.g., appointment conflicts)** require integration testing rather than unit testing. |
| **Database Testing** | Uses an **in-memory database** for testing CRUD operations without affecting production data. | Does not fully replace **real-world testing with live patient records**, requiring additional validation. |
| **Security & Role Testing** | Strict role-based access testing prevents **unauthorized actions and data leaks**. | Does not **fully automate penetration testing**, which must be performed separately. |
| **Continuous Integration** | Runs automated regression tests after every code update to **catch issues early**. | Large test suites can **increase execution time**, making it necessary to optimize the test cases. |

### 

### What Medicure Should Implement

* **Isolated Testing Without External Dependencies** Structure the application to allow testing of core logic without relying on real databases or authentication services. This helps keep unit tests fast and focused.
* **Dummy txt FIle Testing for CRUD Operations** Use a dummy text files during testing to verify that Create, Read, Update, and Delete operations maintain data consistency.
* **Role-Based Access Control (RBAC) Testing** Write tests to ensure that users with different roles (e.g., Admin, Doctor, Patient) can only perform actions permitted to their role, helping prevent unauthorized access or modifications.
* **Automated Regression Testing with MSTest** Implement a suite of MSTest-based regression tests to verify that updates and changes do not unintentionally break existing functionality.

### What Medicure Can Improve

* **Adopt Testing Best Practices from Other Systems** While OpenMRS is based on Java and Spring Boot, Medicure can adapt similar testing principles and structures using Winforms and MSTest.
* **Test Categorization for Efficiency** Improve test execution efficiency by logically separating unit tests (which run quickly and test isolated functions) from integration tests (which validate end-to-end behavior).
* **Potential for Future UI Test Automation** Although not implemented in the current version, UI testing can be explored in the future using frameworks compatible with WinForms to expand coverage beyond logic-level testing.

## 1.2. ClinicSys – Comprehensive Clinic Management System

### Overview

**ClinicSys** is a **clinic management software** designed to **streamline medical workflows** in private clinics, small healthcare facilities, and specialty hospitals. It provides a **cloud-based** and **on-premises** solution for medical professionals to manage **patient data, appointments, billing, and administrative tasks efficiently**.

##### Key Features of ClinicSys

* **Patient Record Management** – Digital storage of patient demographics, medical history, allergies, and prescriptions.
* **Appointment Scheduling** – Automated doctor-patient scheduling with real-time availability.
* **Billing & Payments** – Handles invoices, medical billing, and insurance claim processing.
* **Inventory Management** – Tracks the availability of medicines and medical supplies.
* **Role-Based Access Control (RBAC)** – Ensures secure access control for **doctors, receptionists, and administrators**.
* **Telemedicine & Remote Consultations** – Supports **video consultations and online prescription management**.

Since **Medicure** shares similar **clinic management functionalities**, **studying ClinicSys' testing methodologies** will help refine our **unit testing strategies in MSTest**.

### Unit Testing Approach in ClinicSys

ClinicSys is primarily built using **C# and ASP.NET Core**, making it a **directly relevant case study for Medicure**. It leverages **MSTest, xUnit, and Moq** for unit testing and follows a **modular testing approach**.

#### 1. Mocking Dependencies

ClinicSys interacts with multiple **external services**, including:

* **Insurance providers** for claim validation.
* **Laboratory APIs** for test results.
* **Payment gateways** for billing transactions.

To isolate unit tests from external dependencies, ClinicSys uses **Moq to create mock objects** for:

* Simulating **database queries** without affecting the actual database.
* Testing **external API responses** without requiring real API calls.
* Ensuring **secure authentication flow** without hitting real user credentials.

##### How This Relates to Medicure

* **Medicure** also integrates **database interactions, authentication, and appointment scheduling**.
* Using **Moq in MSTest**, we can create **mock objects** for services such as:
  + **Mocking database calls** to test patient record retrieval.
  + **Mocking authentication services** to verify secure login behavior.

#### 2. Program Logic Testing

**linicSys** uses a relational database to manage critical data like patient history, appointments, and billing. Its unit testing approach includes:

* SQLite In-Memory testing for CRUD operations
* Transaction rollback tests for consistency
* Index and query performance evaluations

**How This Relates to Medicure**

While Medicure does not use a traditional database, it maintains persistent data using structured .txt files. Therefore, its testing strategy should be adapted accordingly:

* **Simulate File-Based CRUD Testing** Implement unit tests in MSTest to validate Create, Read, Update, and Delete operations on text-based records. Use temporary files or in-memory data structures during testing to avoid modifying real patient data.
* **Ensure File Consistency** Verify that read/write operations maintain consistent file formats and do not introduce data corruption, especially during updates and deletions.
* **Optimize File Access** Test file read and write operations to ensure they are efficient and scalable for larger datasets, as Medicure grows in usage.

#### 3. Authentication & Role-Based Access Testing

ClinicSys enforces **strict role-based authentication**, ensuring:

* **Doctors** can access patient history but not modify financial records.
* **Receptionists** can manage appointments but not access medical records.
* **Administrators** can add/remove staff but cannot alter prescriptions.

To test this, ClinicSys:

* Uses **unit tests to validate access rights** for different user roles
* Performs **security penetration testing** for unauthorized access attempts.

##### How This Relates to Medicure

* Medicure also requires **RBAC enforcement**.
* We will write **unit tests in MSTest** to validate:
  + **Unauthorized users cannot modify medical data.**
  + **Doctors have restricted access to financial records.**
  + **Admins can manage users but not alter patient prescriptions.**

### Strengths and Weaknesses of ClinicSys’ Testing Approach

| **Factor** | **Strengths** | **Weaknesses** |
| --- | --- | --- |
| **Testing Framework** | Uses **MSTest and xUnit** for comprehensive testing. | Requires **large test suites**, which can slow down development speed. |
| **Mocking & Isolation** | Implements **Moq to simulate external services**. | Some **edge cases require live API testing**, which cannot be fully mocked. |
| **Security & Role Testing** | Follows **RBAC policies** for strict access control. | Requires **manual security audits** beyond unit testing. |
| **Continuous Integration** | Runs **automated regression tests** for stability. | Requires **test execution optimization** to avoid performance bottlenecks. |

### What Medicure Should Implement

* **Text File-Based CRUD Testing with MSTest** Adapt traditional database test strategies to the current file-based storage system by testing Create, Read, Update, and Delete operations on structured .txt files using MSTest.
* **RBAC Authentication Testing** Implement unit tests to validate role-based access control, ensuring that each user type (e.g., Admin, Doctor, Patient) has appropriate access to the system’s features.
* **Regression Test Automation with MSTest** Maintain a suite of regression tests using MSTest to automatically verify that new updates don’t break existing features or introduce bugs.
* **Foundations for Future CI/CD Integration** Prepare the test structure and scripts for integration into CI/CD pipelines (e.g., GitHub Actions, Azure DevOps) to support automated test execution in future development cycles.

### What Medicure Can Improve

* **Consider Future UI Testing for WinForms** While not currently implemented, end-to-end UI testing using tools like Selenium or WinAppDriver can be explored in future iterations to validate the user interface automatically.
* **Test Execution Optimization** Organize and group test cases by type (e.g., fast logic tests vs. slower file-based tests). Future improvements could include running tests in parallel to reduce overall execution time.
* **Explore Lightweight Performance Testing** Although not using a traditional database, performance testing of file I/O operations (e.g., read/write time for patient records) can help ensure the system remains responsive as data volume grows.

## 1.3. MedKey – Cloud-Based Medical Practice Management

### Overview

**MedKey** is a **cloud-based clinic management system** designed for **multi-specialty hospitals, private clinics, and diagnostic centers**. It provides an **integrated platform** for managing **patient records, medical billing, appointments, and reporting**. MedKey is widely adopted by **healthcare institutions** looking for a **scalable, remote-access solution**.

##### Key Features of MedKey

* **Patient & Appointment Management** – Centralized **patient history tracking** and **scheduling**.
* **E-Prescription System** – Allows doctors to generate, store, and share **digital prescriptions**.
* **Medical Billing & Insurance Handling** – Automates **invoice generation** and **insurance claim processing**.
* **Telemedicine Support** – Facilitates **video consultations** and remote diagnostics.
* **Integrated Lab Reporting** – Directly connects with **diagnostic labs** for **test result updates**.
* **User Role Management** – Ensures **secure access** for different user levels (doctors, patients, and admin staff).

Since **MedKey aligns with Medicure’s core functionalities**, its **testing strategies** provide **valuable insights** for implementing **MSTest-driven validation** in Medicure.

### Unit Testing Approach in MedKey

MedKey is built using **ASP.NET Core with C#** and follows a **microservices-based architecture**. It leverages **MSTest, xUnit, and Moq** for robust testing.

#### 1. Mocking Dependencies

MedKey interacts with **third-party healthcare services**, including:

* **Pharmacy APIs** for digital prescriptions.
* **Insurance APIs** for claim validation.
* **Telemedicine platforms** for remote consultations.

To ensure **unit tests remain isolated**, MedKey uses **Moq** for:

* **Simulating API calls** for real-time insurance claim validation.
* **Mocking external pharmacy integrations** without actual API calls.
* **Testing authentication services** without exposing real credentials.

##### How This Relates to Medicure

Although Medicure does not currently integrate with external APIs, it does include components like authentication and billing logic that could benefit from dependency isolation during testing.

* Current Implementation: Medicure uses MSTest without mocking libraries. Tests are focused on internal logic, using controlled test data instead of external services.
* Future Potential: As Medicure evolves, integrating a mocking library like Moq can help simulate API responses, making it easier to test authentication, billing, or other service interactions without relying on real endpoints.

This approach would allow for faster, more reliable tests while maintaining clean separation between business logic and external systems.

#### 

#### 2. Authentication & Role-Based Access Testing

MedKey enforces **multi-layered authentication** and **role-based access controls**, ensuring:

* **Doctors** can update patient records but not modify financial transactions.
* **Administrators** can manage **staff roles** but not alter **medical prescriptions**.
* **Patients** can view their own history but **cannot access other users' data**.

To validate this, MedKey:  
 **Implements unit tests** for unauthorized access prevention.  
 **Uses integration testing** to verify API authentication flows.

##### How This Relates to Medicure

* **Medicure requires similar role-based authentication mechanisms**, making **MSTest-based security validation critical**.
* **Unit tests will validate role permissions** to ensure correct **access restrictions**.

### 

### Strengths and Weaknesses of MedKey’s Testing Approach

| **Factor** | **Strengths** | **Weaknesses** |
| --- | --- | --- |
| **Testing Framework** | Uses **MSTest** for comprehensive unit testing. | **Microservices testing requires more integration tests**, increasing complexity. |
| **Mocking & Isolation** | Uses **Moq to simulate API calls and services**. | Some **API behaviors cannot be fully replicated**, requiring live testing. |
| **Security & Role Testing** | Implements **multi-layered authentication and RBAC validation**. | Requires **additional penetration testing** for **advanced security threats**. |
| **Continuous Integration** | Uses **automated regression testing** via **Azure DevOps**. | Large test suites **may slow down development pipeline execution**. |

### 

### What Medicure Should Implement

* **Isolated Logic Testing Using MSTest** Continue validating business logic independently from storage and external services using MSTest, ensuring stable and repeatable test execution.
* **File-Based CRUD Testing** Simulate database-like testing by validating CRUD operations on structured .txt files, ensuring consistency without using a relational database.
* **Role-Based Access Control (RBAC) Testing** Write unit tests in MSTest to verify that users are restricted based on roles (e.g., only Admins can delete records), strengthening access security.
* **Prepare for CI/CD Automation** Structure test cases and output reports in a format compatible with CI/CD platforms (e.g., GitHub Actions, Azure Pipelines), setting the groundwork for automated testing workflows.

### **What Medicure Can Improve**

* **Plan for Future Use of Moq for External Services** Although currently not integrated, using Moq in the future can allow simulation of authentication or billing services, making tests faster and more focused on core logic.
* **Explore WinForms UI Automation** Consider adding UI testing support using tools like **WinAppDriver** or **Selenium with WinForms bridges** to automate and validate UI flows.
* **Improve Test Performance with Parallel Execution** Organize tests by scope and enable future support for parallel test execution to speed up validation as the test suite grows.
* **Explore Lightweight Performance Monitoring** Begin benchmarking file read/write operations under load to monitor efficiency and responsiveness as the system scales.