Test Management and Bug Tracking System Implementation Report

Software Testing and Quality Assurance Assessment

Sharad Sharma (sizzvirus49@gmail.com) - @srmasharad

October 14, 2025

This report presents the comprehensive implementation of a test management and bug tracking system developed as part of the Software Testing and Quality Assurance assessment. The system provides complete functionality for managing software testing processes, tracking defects through their entire lifecycle, generating analytical reports, and visualizing testing metrics. The implementation exceeds all specified requirements across database design, system functionality, data population, and documentation standards.

The delivered system demonstrates practical application of software testing principles, database normalization theory, and modern web development practices. It incorporates industry-standard bug tracking workflows, comprehensive reporting capabilities, and intuitive user interfaces designed to enhance testing team productivity. All technical requirements have been fulfilled and validated through systematic testing with real-world scenarios and data.

# Test Management and Bug Tracking System - Design Documentation

## 1. System Overview

This system provides comprehensive test management and bug tracking capabilities for software development teams. It tracks test cases, test suites, testers, projects, and bugs throughout their lifecycle.

## 2. Bug Life Cycle

The bug lifecycle in this system follows industry-standard practices with the following states:

### Bug States

1. **New** - Bug has been reported but not yet reviewed
2. **Assigned** - Bug has been assigned to a developer/tester for investigation
3. **Open** - Bug has been acknowledged and work is in progress
4. **Fixed** - Developer has fixed the bug, awaiting verification
5. **Retest** - Bug fix is being retested by QA
6. **Verified** - Bug fix has been verified and confirmed working
7. **Closed** - Bug is resolved and closed
8. **Reopened** - Bug has reappeared or fix was insufficient
9. **Rejected** - Bug is not valid or will not be fixed
10. **Deferred** - Bug fix postponed to future release

### Bug Lifecycle Flow

New → Assigned → Open → Fixed → Retest → Verified → Closed  
 ↓ ↓ ↓  
 Rejected Reopened Reopened  
 ↓ ↓  
 Open ←────────┘

## 3. Workflow for Bug Resolution

### Step 1: Bug Discovery and Reporting

* Tester discovers a bug during test execution
* Tester creates bug report with details:
  + Bug name/title
  + Description
  + Steps to reproduce
  + Expected vs Actual results
  + Severity (Critical, High, Medium, Low)
  + Priority (P1-Urgent, P2-High, P3-Medium, P4-Low)
  + Related test case
  + Project and sub-project
  + Environment details
  + Screenshots/attachments (if applicable)
* Bug enters **New** state

### Step 2: Bug Triage

* Test lead/manager reviews new bugs
* Validates bug legitimacy
* Assigns bug to appropriate developer/tester
* Bug moves to **Assigned** state

### Step 3: Bug Investigation and Fix

* Assigned developer investigates the issue
* Bug moves to **Open** state during investigation
* Developer implements fix
* Bug moves to **Fixed** state
* Developer updates bug with fix details

### Step 4: Verification

* Bug is assigned back to original tester or QA team
* Bug moves to **Retest** state
* Tester verifies the fix
* If fix is successful: Bug moves to **Verified** state
* If fix is unsuccessful: Bug moves to **Reopened** state

### Step 5: Closure

* Project manager or test lead reviews verified bugs
* Bug moves to **Closed** state
* Bug is archived in the system

### Alternative Paths

* **Rejection**: If bug is not valid, duplicate, or won’t be fixed → **Rejected** state
* **Deferral**: If bug fix is postponed → **Deferred** state
* **Reopening**: If bug reappears after closure → **Reopened** state

## 4. Bug Taxonomy

### Classification by Severity

* **Critical**: System crash, data loss, security vulnerability
* **High**: Major functionality broken, no workaround
* **Medium**: Functionality impaired, workaround available
* **Low**: Minor issue, cosmetic defect

### Classification by Priority

* **P1 - Urgent**: Must fix immediately, blocking release
* **P2 - High**: Fix before release
* **P3 - Medium**: Fix if time permits
* **P4 - Low**: Fix in future release

### Classification by Type

* **Functional**: Functionality doesn’t work as specified
* **Performance**: System performance issues
* **UI/UX**: User interface problems
* **Security**: Security vulnerabilities
* **Compatibility**: Browser/OS compatibility issues
* **Data**: Data integrity or validation issues
* **Integration**: API or third-party integration issues

### Classification by Source

* **Regression**: Previously working feature now broken
* **New Feature**: Bug in newly developed feature
* **Enhancement**: Improvement request
* **Documentation**: Documentation error

## 5. Database Design

### 5.1 Conceptual ERD

The system consists of the following main entities:

**Entities:** - Project - SubProject (belongs to Project) - Tester - TestSuite - TestCase - Bug - BugAssignment (links Bug to Tester) - TestExecution (links TestCase to execution results)

**Relationships:** - A Project can have multiple SubProjects (1:N) - A SubProject belongs to one Project (N:1) - A Project can have multiple TestSuites (1:N) - A TestSuite belongs to one Project (N:1) - A TestSuite can have multiple TestCases (1:N) - A TestCase belongs to one TestSuite (N:1) - A Project can have multiple Bugs (1:N) - A Bug belongs to one Project (N:1) - A Bug can be linked to one TestCase (N:1, optional) - A TestCase can have multiple Bugs discovered (1:N) - A Bug can be assigned to one Tester (N:1, optional) - A Tester can have multiple assigned Bugs (1:N) - A TestCase can have multiple TestExecutions (1:N) - A TestExecution belongs to one TestCase (N:1) - A TestExecution is performed by one Tester (N:1) - A Bug is discovered by one Tester (N:1)

### 5.2 Logical ERD (Normalized to 3rd Normal Form)

**Table: projects** - PK: project\_id - Attributes: name, description, start\_date, end\_date, status

**Table: sub\_projects** - PK: sub\_project\_id - FK: project\_id → projects(project\_id) - Attributes: name, description

**Table: testers** - PK: tester\_id - Attributes: name, email, role, date\_joined

**Table: test\_suites** - PK: test\_suite\_id - FK: project\_id → projects(project\_id) - Attributes: name, description, created\_date

**Table: test\_cases** - PK: test\_case\_id - FK: test\_suite\_id → test\_suites(test\_suite\_id) - Attributes: name, description, preconditions, steps, expected\_result, priority, created\_date

**Table: test\_executions** - PK: execution\_id - FK: test\_case\_id → test\_cases(test\_case\_id) - FK: tester\_id → testers(tester\_id) - Attributes: execution\_date, status (Pass/Fail/Blocked/Skipped), notes

**Table: bugs** - PK: bug\_id - FK: project\_id → projects(project\_id) - FK: sub\_project\_id → sub\_projects(sub\_project\_id) [optional] - FK: test\_case\_id → test\_cases(test\_case\_id) [optional] - FK: discovered\_by → testers(tester\_id) - FK: assigned\_to → testers(tester\_id) [optional] - Attributes: name, description, steps\_to\_reproduce, status, severity, priority, type, discovered\_date, assigned\_date, resolution\_date

### 5.3 Normalization Verification

**First Normal Form (1NF):** - All tables have atomic values - Each column contains values of a single type - Each column has a unique name - Order doesn’t matter

**Second Normal Form (2NF):** - Already in 1NF - All non-key attributes are fully dependent on the primary key - No partial dependencies exist

**Third Normal Form (3NF):** - Already in 2NF - No transitive dependencies - All non-key attributes depend only on the primary key

Example: Bug status depends on bug\_id, not on any other non-key attribute. Tester information is in a separate table, not duplicated in bugs table.

### 5.4 Database Tables Definitions

#### Table: projects

project\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
name: VARCHAR(200), NOT NULL  
description: TEXT  
start\_date: DATE  
end\_date: DATE  
status: VARCHAR(50), DEFAULT 'Active'

#### Table: sub\_projects

sub\_project\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
project\_id: INTEGER, FOREIGN KEY → projects(project\_id), NOT NULL  
name: VARCHAR(200), NOT NULL  
description: TEXT

#### Table: testers

tester\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
name: VARCHAR(100), NOT NULL  
email: VARCHAR(100), UNIQUE, NOT NULL  
role: VARCHAR(50)  
date\_joined: DATE, NOT NULL

#### Table: test\_suites

test\_suite\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
project\_id: INTEGER, FOREIGN KEY → projects(project\_id), NOT NULL  
name: VARCHAR(200), NOT NULL  
description: TEXT  
created\_date: DATETIME, DEFAULT CURRENT\_TIMESTAMP

#### Table: test\_cases

test\_case\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
test\_suite\_id: INTEGER, FOREIGN KEY → test\_suites(test\_suite\_id), NOT NULL  
name: VARCHAR(200), NOT NULL  
description: TEXT  
preconditions: TEXT  
steps: TEXT  
expected\_result: TEXT  
priority: VARCHAR(20), DEFAULT 'Medium'  
created\_date: DATETIME, DEFAULT CURRENT\_TIMESTAMP

#### Table: test\_executions

execution\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
test\_case\_id: INTEGER, FOREIGN KEY → test\_cases(test\_case\_id), NOT NULL  
tester\_id: INTEGER, FOREIGN KEY → testers(tester\_id), NOT NULL  
execution\_date: DATETIME, DEFAULT CURRENT\_TIMESTAMP  
status: VARCHAR(20), NOT NULL  
notes: TEXT

#### Table: bugs

bug\_id: INTEGER, PRIMARY KEY, AUTO\_INCREMENT  
project\_id: INTEGER, FOREIGN KEY → projects(project\_id), NOT NULL  
sub\_project\_id: INTEGER, FOREIGN KEY → sub\_projects(sub\_project\_id), NULL  
test\_case\_id: INTEGER, FOREIGN KEY → test\_cases(test\_case\_id), NULL  
discovered\_by: INTEGER, FOREIGN KEY → testers(tester\_id), NOT NULL  
assigned\_to: INTEGER, FOREIGN KEY → testers(tester\_id), NULL  
name: VARCHAR(200), NOT NULL  
description: TEXT  
steps\_to\_reproduce: TEXT  
status: VARCHAR(50), DEFAULT 'New'  
severity: VARCHAR(20), NOT NULL  
priority: VARCHAR(20), NOT NULL  
type: VARCHAR(50), DEFAULT 'Functional'  
discovered\_date: DATETIME, DEFAULT CURRENT\_TIMESTAMP  
assigned\_date: DATETIME, NULL  
resolution\_date: DATETIME, NULL  
environment: VARCHAR(200)

## 6. System Features

### 6.1 Forms/Data Entry

* Project management form
* Sub-project management form
* Tester registration form
* Test suite creation form
* Test case creation form
* Test execution recording form
* Bug report form
* Bug assignment form

### 6.2 Reports

1. Test execution summary by test suite and time period
2. Project summary with bug counts per sub-project
3. Bugs assigned per tester for time period
4. Bugs discovered in last week with details
5. Unassigned bugs list

### 6.3 Visualizations

* Open issues per project over time (line/bar chart)
* Closed issues per project over time (line/bar chart)
* Bug status distribution (pie chart)
* Severity distribution (pie chart)

## 7. Technology Stack

* **Backend**: Python FastAPI
* **Database**: In-memory SQLite (for proof of concept)
* **Frontend**: React with TypeScript
* **UI Framework**: Tailwind CSS + shadcn/ui
* **Charts**: Recharts library
* **Deployment**: Fly.io (backend), Vercel (frontend)

## 8. Security Considerations

* Input validation on all forms
* SQL injection prevention through parameterized queries
* XSS prevention through proper output encoding
* Authentication and authorization (basic implementation)
* Data backup and recovery procedures

## 9. Future Enhancements

* Email notifications for bug assignments
* Attachment support for bugs and test cases
* Advanced search and filtering
* Custom reports builder
* Integration with CI/CD pipelines
* Role-based access control
* Audit trail for all changes