# Mekelle_University__crest_-removebg-preview

**STUDENT REGISTRATION SYSTEM**

**TESTING PLANNING, ANALYSIS AND DESIGN**

Rozina Gebrehiwet

Eitm/ur170202/12

Jan 16, 2025 G.C.

[1. Testing Planning and Control 3](#_Toc21072)

[1.1 Test Planning 3](#_Toc32485)

[1.1.1 Objective 3](#_Toc6261)

[1.1.2 Components of a Test Plan 3](#_Toc31114)

[1.1.3 Tools for Test Planning 8](#_Toc27124)

[1.2 Test Control 8](#_Toc18156)

[1.2.1 Objectives 8](#_Toc10704)

[1.2.2 Activities 8](#_Toc18029)

[1.2.3 Tools for Test Control 10](#_Toc18601)

[1.2.4 Deliverables 11](#_Toc489)

[Test Metrics Reports 11](#_Toc32192)

[Final Test Report 11](#_Toc7462)

[2. Test Analysis and Design 12](#_Toc10529)

[2.1 Test Analysis 12](#_Toc7644)

[2.1.1 Objectives 12](#_Toc12313)

[2.1.2 Activities 12](#_Toc14540)

[2.2 Test Design 14](#_Toc28257)

[2.2.1 Objectives 14](#_Toc29891)

[2.2.2 Activities 14](#_Toc27050)

[2.2.3 Deliverables 15](#_Toc14971)

[2.2.4 Tools for Analysis and Design 15](#_Toc8808)

# Testing Planning and Control

## Test Planning

### Objective

The objective is to guarantee the quality and dependability of the Student Registration System, establish clear boundaries for testing, and allocate resources in an effective manner.

### Components of a Test Plan

#### Introduction

Intended to streamline student registration, information and having a simple experience of use. This system is primarily for testing the functionality, reliability and usability of this, and to find any defect in the whole process. There would be key stakeholders involved in the process namely

1. developer creating the system that would be building
2. testers for finding defects

#### Scope of Testing

Primary testing areas would be the user registration process integration with Chapa (suceeds and fails both), admin dashboard functionality to sustain/manage student data basically. Moreover, user authentication and the upholding of security for sensitive data are essential for rigorous evaluation. Response time, load handling are also very important non-functional aspects in order to give a robust system which can be used correctly and efficiently. However, the testing of features that are beyond Chapa's response handling with regards third party APIs is skipped because those have been subjected to independent testings.

#### Testing Objectives

The primary goals are to confirm the proper functioning of the user registration process and to verify that payment processing systems operate securely and dependably.

#### Test Strategy

**Testing Levels:**

**Unit Testing** – form validations, API call etc, individual functions

**Integration Testing:** To check if the Django views are talking with the Chapa module smoothly.

**System Testing:** verify complete registration to pay workflow.

**Acceptance testing:** Compliance with requirements to be tested by user feedbacks

**Testing Types:**

**Manual Testing** – Scenarios like innovative and critical path testing

**Automation Testing:** Use Selenium for UI flows, and use Pytest for backend code. **Load Testing:** Ensure that system can concurrently handle multiple simultaneous registrations and payments being processed.

**Test Environment Requirements:** Staging env similar to Prod for database testing and payment APIs.

#### Test Deliverables

**Test Cases and Scripts**

| **Test Case ID** | **Scenario** | **Description** | **Preconditions** | **Steps** | **Expected Results** | **Priority** |
| --- | --- | --- | --- | --- | --- | --- |
| **TC\_REG\_001** | Invalid Email Validation | Verify error handling for invalid email inputs. | Registration form is displayed with email validation logic in place. | 1. Open registration form.  2. Enter valid first name/last name but invalid email format.  3. Click "Register". | Error displayed: "Please enter a valid email address."  Registration fails. | High |
| **TC\_REG\_002** | Missing Fields in Registration | Test form handling when mandatory fields are not provided. | Registration form is displayed. | 1. Open registration form.  2. Leave one or more mandatory fields empty.  3. Click "Register". | System shows error highlighting empty fields.  Error message: "This field is required." | Medium |
| **TC\_PAY\_001** | Successful Payment via Chapa | Verify Chapa processes payment successfully. | Student is logged in; Chapa sandbox environment is ready. | 1. Login as student.  2. Fill valid registration details   3.Click ‘Register’ button 4. Enter valid payment details via Chapa.  4. Confirm payment. | Payment succeeds.  Success message displayed.  Receipt email sent. | High |
| **TC\_PAY\_002** | Payment Failure with Invalid input field | Ensure system handles payment failure gracefully for invalid input details. | Student is logged in; Chapa sandbox is ready. | 1. Login as student.  2.Fil registration details.   3. Click ‘register’button. 4. Enter invalid input details in Chapa.  4. Confirm payment. | Payment fails.  Error: "Transaction declined. Use a valid payment method." | High |
| **TC\_PAY\_003** | Payment Failure with Network Interruption | Test system response when network connection is lost during payment. | Student is logged in and proceeds to payment page; network can be interrupted. | 1. Start payment process.  2. Disable network connection during transaction.  3. Observe behavior. | Payment fails gracefully.  Message: "Network error. Please try again."  No amount deducted. | Medium |
| **TC\_SEC\_001** | HTTPS Payment Security | Validate payment transactions use HTTPS for secure communication. | Payment page is displayed; Browser displays URL and SSL certificate details. | 1. Access payment page.  2. Check URL starts with https://.  3. Verify SSL certificate is valid. | Payment communication is secured via HTTPS. | High |
| **TC\_PAY\_004** | Duplicate Payment Attempts | Ensure the system prevents duplicate payments for the same registration. | Student is logged in and has already successfully paid for the course. | 1. Login as student.  2. Attempt to pay again for the same course. | Duplicate payment is blocked.  Message: "Payment for this course already made." | Medium |

**Test Data**

Types of Data: valid Email, invalid email , payment Entrance (valid and invalid situations)

**Test Reports**

Key Statistics: pass rates/Fail rates

**Defect Logs**

Categorized as: Severity levels (high, medium, low)

#### Entry and Exit Criteria

Entry Criteria

1. ****Requirements Documentation Final —**** All requirements documentation should be finalized, reviewed.
2. ****Development of all Modules Completed****: Proper development of all modules must be in 100% and they are ready for testing.
3. Setup Test Environment with Chapa Sandbox: A standalone test environment should be already created and must verify all Chapa integration totally functional.

Exit Criteria

1. ****Fix all high and critical severity defects****: Any defect identified to be of high or critical severeness during testing must get fixed and verified as closed.
2. ****Functional Coverage Test at Least 95% :**** You need to ensure that the functional tests are at least testing 95% of the required requirements and functions.

#### Resources

****Team Size****: with one classmate.

**Tools and Systems**

****Tools****:

* Selenium: Functional & regression tool for automation.
* Python Strains: a Framework for both unit and API testing.
* Postman: Systems to check an API temporarily.

****Systems:**** Desktops supplied, having access to live servers for running and monitoring tests expeditiously.

#### Schedule

| **Task** | **Start Date** | **End Date** | **Responsible** |
| --- | --- | --- | --- |
| Test case creation | Jan 14, 2025 | Jan 17, 2025 | Dev |
| Environment setup | Jan 14, 2025 | Jan 17, 2025 | Dev |
| Execution of tests | Jan 21, 2025 | Jan 26, 2025 | Team mate |
| Defect logging/fixing | Jan 20, 2025 | Jan 30, 2025 | Dev |
| Final test report | Jan 31, 2025 | Jan 31, 2025 | Dev |

#### Risk Management

****Risks****:

1. Chapa API DownTime, outages can impact payment testing.
2. Loss of Transaction Data in case of Network Disruption: Data may be compromised in a network that is not working stably.

**Strategies for Mitigation:**

1. ****Build Fallbacks for the API Tests****: Build & robustly test fallback APIs so you don't need break internal functionality when an API is down.

****B. Continuous Backups of Sensitive User Information:**** Run backups frequently so no data is lost and the user can be restored as soon as their network comes.

#### Approval

Test plan and results to be reviewed and approved by instructor Messele

### Tools for Test Planning

**Documentation Tools**: Microsoft Word for writing the test plan.

**Test Management Tools**: TestRail for managing test cases, tracking execution, and reporting results.

**Version Control Tools**: Github to ensure test scripts and documentation are appropriately versioned.

## Test Control

### Objectives

1. ****Complies to the Test Plan****: Make sure testing processes stick to the established test plan.
2. ****Flexibility****: Ensures that the schedule is flexible to factor any changes in the scope of a project and user requirements.
3. ****Transparent with the stakeholders****: Ensure transparency on the stakeholder visibility of progress and results for testing so far.

### Activities

##### Monitoring

1. Test Case Execution

**Objective**: Measure the progress of testing activities.

**Metrics**:

Track the number of executed test cases against the planned total.

Calculate the execution percentage to assess progress.

**Action**: If execution is lagging, identify bottlenecks and adjust resources as needed.

1. Defect Trends

**Objective**: Analyze defect patterns to inform quality assurance.

**Metrics**:

Monitor defect density (defects per test case).

Evaluate severity levels of defects to identify critical issues.

**Action**: Use trend analysis to predict potential quality impacts and adjust testing focus accordingly.

1. Schedule Adherence

**Objective**: Ensure testing milestones are met as planned.

**Metrics**:

Track adherence to the testing schedule, including milestone completion dates.

**Action**: Identify any delays early and implement corrective actions to minimize impacts on the overall project timeline.

##### Reporting

Comprehensive Reports

**Objective**: Provide stakeholders with clear insights into testing progress and quality.

**Content**:

* Summary of test progress (executed vs. planned cases).
* Risks identified during testing.
* Metrics such as pass/fail rates and defect counts.

**Frequency**: Prepare reports weekly or at key project milestones.

##### Issue Management

1. Conflict Resolution

**Objective**: Address bottlenecks that may hinder testing progress.

**Actions**:

Escalate issues such as delays in defect fixes to relevant stakeholders.

Facilitate discussions to ensure timely resolution of conflicts.

1. Resource Allocation

**Objective**: Optimize resource use to overcome testing challenges.

**Actions**:

Assess resource needs based on current testing progress.

Redirect team members as necessary to address critical bottlenecks and reduce delays.

##### Change Management

* Dynamic Adaptation

**Objective**: Ensure testing remains aligned with evolving project requirements.

**Actions**:

Revise test plans as needed to accommodate changes in requirements.

Adapt the scope of testing, particularly for critical components like APIs (e.g., Chapa API).

##### Defect Tracking

1. Tracking Tools

**Objective**: Manage defects efficiently throughout the testing lifecycle.

**Tools**: Utilize defect tracking tools such as Jira for effective management.

**Focus Areas:**

Prioritize defects based on severity and impact on the project.

Ensure timely logging of defects and track their resolution status.

1. Priority Assignment

**Objective**: Ensure critical defects are addressed promptly.

**Actions**:

Assign severity levels to defects (e.g., critical, major, minor).

Monitor the progress of defect rectification closely, ensuring that high-priority issues are resolved first.

### Tools for Test Control

##### Test Management

**TestRail**: A comprehensive test management tool that supports test case design, execution, and reporting. Customizable test plans, detailed reporting, and integration with various issue trackers.

##### Defect Tracking

**Jira**: A popular issue tracking tool that supports defect tracking and project management. Custom workflows, reporting capabilities, and integration with various development and testing tools.

##### Dishoarding

**Jira**: A versatile issue tracking tool that supports defect management and prioritization. Custom workflows for defect resolution, backlog management, and integration with test management tools.

### Deliverables

**Test Metrics Reports**

Test Coverage=

(Number of Test Cases Executed/Total Number of Requirements)×100.

Execution Status

Metrics to Report:

Total Test Cases: Number of test cases planned.

Executed Test Cases: Number of test cases executed.

Passed Test Cases: Number of test cases that passed.

Failed Test Cases: Number of test cases that failed.

Blocked Test Cases: Number of test cases that could not be executed due to dependencies.

Defect Statistics

Defect Resolution Rate=(Defects Resolved/Total Defects Logged)×100

**Final Test Report**

**Holistic Summary:** The summary provides an overview of the testing process, including its scope, objectives, and key outcomes such as test execution results, defect statistics, resolution rates, and test coverage percentages.

# Test Analysis and Design

## Test Analysis

### Objectives

- Extract exact test conditions and scenarios that are coming form formally written project requirements.

- Link every test condition to a single requirement for full requirement coverage.

- Review Chapa API and integration docs to see if we are missing any edge cases.

### Activities

This section outlines the key activities, including requirement analysis, test basis identification, deriving test conditions, and prioritization.

##### Requirement Analysis

**Activities**

Analyze Requirements:

**Sources**: Gather requirements from various sources, including:

* User stories
* Technical documentation
* API guides

**Process:**

Review each requirement for clarity and completeness.

Identify any ambiguities or missing information that may impact testing.

Categorize Tests:

**Types of Tests:**

* Functional Testing: Validate that the software behaves as expected.
* Security Testing: Ensure that the application is secure against vulnerabilities.
* Non-Functional Testing: Assess aspects like scalability and performance.
* Implementation: Create a matrix to categorize tests based on these aspects.

##### Test Basis Identification

**Activities**

Use Key Project Artifacts:

* System Design Documents: Analyze documents related to critical workflows, such as Registration workflows, Payment workflows
* Use Cases and Personas: Develop test scenarios based on specific user interactions (e.g., new student registrations with payment).
* Risk Assessment Reports: Review reports that highlight potential risks, such as API integration failures and Security vulnerabilities

##### Deriving Test Conditions

**Activities**

**Define High-Level Scenarios:**

Validation Scenarios:

* Student Registration Process: Test valid and invalid inputs to ensure proper validation.
* Payment Handling: Create scenarios for handling different payment situations, including successful transactions, transaction failure cases (e.g., insufficient funds, network issues)

##### Prioritization

**Activities**

**Assign Testing Priorities:**

* Priority Levels:
* High Priority: Critical functionalities that impact business operations or user experience.
* Medium Priority: Important features that should be tested but are not critical.
* Low Priority: Features that have minimal impact and can be tested if time permits.
* Implementation: Create a priority matrix to visually represent the prioritization of test cases based on their impact and risk assessment.

.

## Test Design

### Objectives

The test design phase aims to ensure comprehensive coverage, validate requirements, identify defects early, and enhance test efficiency by designing reusable test cases and minimizing redundancy, ensuring a smooth testing process.

### Activities

##### Test Case Development:

Create detailed test cases based on the requirements and scenarios identified in the requirement analysis. Each test case should include:

Test Case ID

Test Description

Preconditions

Test Steps

Expected Results

Actual Results (to be filled during execution)

##### Test Data Preparation:

Identify and prepare the necessary test data for executing test cases, including:

* Valid student registration data (e.g., names, emails).
* Invalid data for boundary testing (e.g., incorrect email formats).
* Payment data for testing Chapa integration.

##### Review and Validation:

Conduct peer reviews of the test cases to ensure they meet quality standards and align with requirements.

Validate test cases with stakeholders to confirm coverage and relevance.

##### Test Environment Setup:

Prepare the test environment to ensure it mirrors the production environment.

Set up necessary configurations for the Django application and Chapa payment gateway.

### Deliverables

* Test Case Document: a comprehensive document containing all test cases, organized by functionality and priority.
* Test Data Specification: a document outlining the test data required for executing the test cases, including formats and sources.
* Test Environment Configuration Document: a document detailing the setup of the test environment, including software versions, configurations, and dependencies.
* Review Feedback Document:a summary of feedback received during the review process, including any changes made to the test cases.

### Tools for Analysis and Design

1. **Test Case Management Tool:**

TestRail: For managing and organizing test cases, tracking execution, and reporting results.

1. **Requirement Management Tools:**

Jira: For tracking user stories, acceptance criteria, and linking test cases to requirements.

1. **Version Control Systems:**

Github: For versioning test case documents and ensuring collaborative updates.

1. **Automation Tools (if applicable):**

Selenium: For automating functional tests, especially for the web interface.

Postman: For testing API endpoints, including Chapa integration.