TOME Test Model

The TOME Team
Curtis “Fjord” Hawthorne
Craig Miller
Clint Olson
fREW Schmidt

November 6, 2006

Contents

1 Introduction

1.1 Purpose of Document

The purpose of this document is to specify the testing approach and philosophy for the TOME project.

The major testing categories will be outlined, the testing framework and methodology will be explained, and some sample tests from each category will be listed.

1.2 Testing Philosophy

Any time a change is made to TOME, there is a chance that the change will break some existing functionality. Given the number of subsystems and the variety of critical functions they perform, a standard test set needs to be defined in order to verify TOME's continued integrity after each modification.

Towards this end, an initial set of tests will be constructed to cover three areas:

- Database integrity
- Core TOME module functionality
- Web interface robustness
- Component integration

1.3 Testing Plan

These tests will be run in an automated fashion after any non-cosmetic change to the code base, and will eventually be integrated into a nightly build to be run from the cron scheduler. In addition, as features get added, new tests will be written to cover the new functionality and formalize its presence.

1.4 Bug Handling Procedure

When the tests reveals a bug (or any result that differs from the norm), the results will be automatically emailed to the relevant members of the TOME team for analysis. If a bug is confirmed, it will be entered into the Trac environment and assigned to a team member as part of their weekly action items.

1.5 Risks

Stefan Posthuma once said, ”It is impossible to make anything foolproof because fools are so ingenious.” The principle remains true to this day – despite the best efforts of programmers worldwide, software continues to manifest bugs and behavior that its creators hadn’t thought to test for. True, this problem can be ameliorated to some extent by creating tests based on the specified design requirements of the software and using automation to ensure any bug only happens once, but the possibility will remain to one degree or another no matter what steps are taken.

While it is ever-present, this risk of the unknown is manageable, especially with the aid of automated regression testing and a versioning system (such as Subversion) to roll back changes to a known working build.

2 Test Harness and Automation

2.1 Test Framework

The TOME project will use the Perl module Test::More for its test framework. This module provides an extremely flexible context to specify individual tests in, running the gamut from the simple ”ok” test which evaluates and expression for truth to more advanced expressions like ”is$_d$*eeply*” (*whichcomparesallmembersofadatastructure*)*and*” *like*” (*whichutilizesregularexpressi*

A more complete summary of the Test::More module can be found at the CPAN repository: `http://search.cpan.org/ mschwern/Test-Simple-0.64/lib/Test/More.pm`

2.2 Test Harness

In keeping with the standard operating procedure for Perl projects, the Test::Harness modules will be used for automation. Using the easily-accessible ”prove” command, Test::Harness recursively iterates through each file and directory in the tests folder, running the tests within and compiling the results. Once it finishes, it outputs the results along with a summary section showing how many tests failed, if any.

The command-line nature of Test::Harness makes it ideal for automation with the cron utility, so that even if individual team members forget to run the tests after a change, any problems that have been introduced should be caught when the test suite is triggered by cron as part of the nightly build. The results can then be captured and compiled into a report to be sent to one or all of the team members.

2.3 Example Tests

2.3.1 Database integrity

InterTOME consistency There is a boolean flag in the libraries table that indicates if a floor is eligible for InterTOME loans. This check makes sure that a library cannot be removed from while there are still books reserved by other libraries from that library.

- If a library changes from being in InterTOME to not being in InterTOME, it should give an error if there are any pending reservations from that library.
- Make sure it really checks that the reservations are pending. In other words, if a reservation exists, but has been fulfilled, it should still let you remove the floor.
- If there are no reservations, it should let you change it.

One current semester The semesters table has a boolean flag to mark which semester is "current."

- Ensure that only one semester can be marked as current. Check on both inserts and updates.

3 Component Design

3.1 Pattern Usage

TOME is built around an MVC pattern. The Model is handled by the TOME model, the View is handled by the templates, and the Control is handled by

TOME::Interface.

3.2 Limitations

One major limitation of the current design is that TOME::Interface inherits from TOME. This introduces both namespace pollution and fragile base class issues. However, after assessing the benefits of moving to a delegation approach versus coping with the current approach, it was determined that, for now, we will continue with the existing design, limited as it may be.