TOME Test Model

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

November 13, 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_deeply" (which compares all members of a data structure) and "like" (which utilizes regular expressions to determine the outcome of the test).

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.

TOME Book Removal
Consistency must be maintained when removing books from the database.

- Ensure that books cannot be marked as removed in the tomebooks table if they still have a current checkout.

- Ensure that the book can be removed if either there are no checkouts for that book or if all the checkouts are marked as being checked in.
- Make sure that checkouts of other books don't prevent this one from being removed.

2.3.2 Core TOME module functionality

Patron management
TOME manages patrons through the functions patron_add, patron_update, patron_info, and patron_search. Each of these needs to be tested to ensure compliance with specifications.

- Ensure that patron_search returns the correct value after patron_add is invoked to create a new patron
- Update the patron's info and check to see if the changes were propagated

Checkouts
Exercise the checkout functions of TOME.

- Add a checkout and verify that it exists
- Set a checkout up to fail, then verify that tomebook_can_checkout complains
- Cancel a checkout and verify that it was dropped

Reservations
Exercise the reservation functions of TOME.

- Reserve a book and verify that it is marked as reserved
- Set a reservation up to fail, then verify that tomebook_can_reserve complains
- Fill a reservation and verify that the reservation flag is set to false

2.3.3 Web interface robustness

Graceful fallback
Like any web application, TOME is subject to being fed bad data and malformed requests. When this happens, it needs to fail gracefully, giving the user an informative error message describing the problem.

- Feed the webserver malformed data requests and make sure TOME fails gracefully

Privilege escalation
Since TOME is essentially stateless (due to the nature of web applications), it is possible for malicious users to attempt to gain entrance to sections of TOME they do not have permissions for. In all cases, TOME should refuse them access.

- Verify that attempting to log in with a bad username and password fails
- Check that TOME refuses access to administrative functions when logged in as a non-administrator

2.3.4 Component integration

Basic web-TOME interaction
The web interface and TOME need to interact to a large degree. Even if TOME itself works when called directly, the communication between the web server and TOME could still fail.

- Execute some common functions (adding a book, modifying a user, etc.) via HTTP POST requests and verify that the actions have taken place by calling TOME directly

TOME-PostreSQL interaction
TOME and the database also interact, so some tests will be needed to verify that this is taking place (or not taking place, as the case may be)

- Give TOME bad connection information and verify that it fails
- Execute some common functions in TOME and check to make sure they show up in the database

3 Templates

3.1 Test Results

Test output
Beginning tests...
../tests/db/currsemester....1..3
ok 1 - DB doesn't allow multiple current semesters
ok 2 - But it doesn allow multiple semesters
ok 3 - Can't have multiple current semesters, even on update
ok
../tests/db/username.....1..2
ok 1 - DB won't allow duplicate usernames
ok 2 - But we can add it after we delete the other one
ok
../tests/db/patronemail.....1..2
ok 1 - Database doesn't allow duplicate emails to be inserted
ok 2 - But we can still insert things later
ok
../tests/db/isbnupper.....1..4
ok 1 - Lowercase ISBN does not exist
ok 2 - Uppercase ISBN does exist
ok 3 - Lowercase ISBN doesn't exist, even after update
ok 4 - Uppercase ISBN still exists
ok
All tests successful.
Files=4, Tests=11, 2 wallclock secs (0.58 cusr + 0.35 csys = 0.93 CPU)

3.2 Bug Reporting

Move TOMETest out of the main modules tree
Status: new Reported by: fjord Assigned to: olsonc
Priority: major Milestone: Automated Test Suite

TOMETest should reside somewhere else. It's not really a part of the production modules code.