GSOC10 - Add Unit Testing to Dspace

Add Unit Testing to Dspace - Pere Villega

Dspace currently lacks unit testing, which harms the development of the platform and makes easier to reintroduce past bugs while developing. This project is a proposal to add a set of unit test classes to Dspace, based on JUnit, plus some tools that detect issues in the code so we can improve its quality. - Pere Villega

On Testing

Due to the extension of the subject a new page has been created with more details. Please check Testing for more information on unit testing

Existing approaches

So far there's been some approaches to the problem. As far as we know, the following:

- Elliot Metsger created some unit testing framework as shown at http://presentations.dlpe.gatech.edu/proed/or09/or09_052009_4/index.html
- Scott Phillips did a set of functional tests for DSpace 1.5 as seen at http://scott.phillips.name/2010/04/dspace-functional-tests/
- Gareth Waller has built a set of Selenium tests as commented recently in the developer's list
- Aaron Zeckoski's effort added JUnit-based testing into DSpace Services e.g. http://scm.dspace.org/svn/repo/modules/dspace-services/tags/dspace-services-2.0.1/impl/src/test/java/org/dspace/test/DSpaceAbstractKernelTest.java

Next there is a table that compares these and some other approaches to the problem:

Tools	Type	Notes
JUnit + JMock	Unit Testing	Dspace is tightly integrated with the database (see notes above) which complicates the task of creating a unit test
JUnit + HTMLUnit	Functional	Uses a embedded webserver to run Dspace and runs the tests, using ant, against this instance
Selenium	Functional	Can be run against any running Dspace instance and using several browsers
JUnit + ContiPerf	Unit Testing + Performance Testing	Suffers from the same issues as other Unit Tests, tight integration with database

As we see we have two main approaches:

- Unit testing: automated, suffers from tight integration with database. Refactoring of code would be advisable, but it can be avoided by using Mocks. Tests at low level, just ensuring a class is correct, not checking the whole functionality. Can allow performance testing via ContiPerf.

- Functional testing: doesn't suffer from database integration. Ensure the functionality works, but it may not account for certain situations that would raise errors due to code bugs.

Both would benefit DSpace, as unit testing would ensure code quality while functional testing would ensure the application behaves as expected with standard usage.

Proposals and Scope

The original proposal for the project were to:

- Create a framework for Functional Tests, Integration Tests, Unit Tests and Performance Tests that run automatically on build (when possible)
    - Create all the scaffolding required (mockups, database connections, configs, etc) for the tests to be run

- Integrate with a Continuous Integration server
- (Optional but recommended) Integrate with a code quality management tool
    - I will integrate Cobertura to create reports on the testing coverage

The proposed scope of the project was to apply the tests to the API core. Once done, work would be extended in the remaining time to Manakin and then other subprojects (Sword, LNI, OIA). JSPUI will be left for last, as is an interface that will disappear in the future.

Project Plan

A work plan has been decided in an IRC meeting on May 26th, 2010 (log: [http://duraspace.org/irclogs/index.php?date=2010-05-26|http://duraspace.org/irclogs/index.php?date=2010-05-26]). The work has been divided in 3 parts:

1. Single class Unit Tests. Ideal candidates would be classes such a DCDate. These can be easily tested in isolation. This sort of test, along with an appropriate way for maven to execute them will give the project, and DSpace, and early win. Hopefully it will encourage further tests to be created by other developers, and we could move towards an expectation of new code requiring tests like this before they are committed.
2. Integration Testing, which requires a database to be present. This would ideally work with an embedded database such as hqsql or derby. We need to think about this a little more to decide how best to do this, and how it would interact with the new services architecture.
3. Functional Testing using a tool such as selenium.

It's been decided they will be done in the same order. The aim is not to do tests for all the existing classes, but to create an infrastructure the community can benefit from and can expand with its own tests.

The first two points will be finished before the mid-term evaluation. The third point will be done after that.

Integration with a continuous integration environment will be possible out-of-the-box,a s we will use Maven for points 1 and 2 and Ant for point 3. The capability of running the functional tests in the continuous integration server depends on configuration as it will require running ant tasks created after the generation of code.

Integration with Cobertura and similar is left to the usage of a quality management tool, as the current state of the Maven plugins creates issues when trying to merge reports from different sub-projects.

What has been Implemented

At the end of the project the following components have been implemented:

- Structure to allow the implementation of unit tests
- Structure to allow the implementation of integration tests
  - Integration tests allows for performance checks
- Unit tests for package org.dspace.content, usable as example
- Two Integration tests usable as example to create new ones
- Added some static analysis tools (like FindBugs) to the build process

No integration with a Continuous Integration environment or a Quality Management tool has been done. This is left for a later phase after the end of GSoC

The Unit Tests created (due to complications during development) have been restricted to those in package org.dspace.content.

Technical description

A tecnical description of the tools used and steps followed is available at GSOC 2010 Unit Tests - Technical documentation

Considerations

DSpace code suffers from testability issues. In the long term it might be advisable to refactor it. Refactoring is dangerous right now as we don't have any test and we risk introducing new bugs to the code base, so it should be avoided until we have enough tests working that makes us confidant we can take that road.

Source Code

The code is held in an SVN branch: http://scm.dspace.org/svn/repo/sandbox/gsoc/2010/testing/

Thanks

This page has been created with help from Stuart Lewis, Scott Phillips and Gareth Waller. I want to thank them all for their comments. Some information has been taken from Wikipedia to make the text more complete. I'm to blame for errors in the text.

Feel free to contribute to this page!