Introduction

Background
The primary goal for this Architecture, Development, and Deployment of a Knowledge Repository and Service is to address the needs of this extended community for a scalable, decentralized infrastructure for managing and disseminating operational metadata and information models, and their associated semantic constructs. The vision for the project is to re-imagine the caBIG technology environment as a more open and more readily extensible framework, one that can grow with less dependency on the centralized processes and systems that are manifest in the first generation of caBIG technology. In particular, the role of the central metadata registry, the caDSR, must be redefined as a federation of metadata registries that can be instantiated and plugged into the caBIG grid or an extended community “cloud” by any qualified entity.

The caDSR has a suite of tools and APIs that support workflows for metadata development, browsing and retrieval. In addition, the caDSR has been adapted to support the UML model-driven development paradigm adopted by caBIG. UML-defined information models such as those from the BRIDG project, caArray, caTissue, and others are each registered in the caDSR through conversion of the model elements into ISO11179 metadata constructs. This functionality, and the workflows that it supports, has evolved over an 8-year period and is now quite mature. It satisfies the current requirements for semantic representation in the current caBIG developer and user community, but it is ill-suited to serve the new requirements for decentralization and indefinite scalability in the broader health care community. The goal of this program is therefore to harvest and recycle the best elements of the first generation of caBIG metadata infrastructure, and to then incorporate those elements into a redesigned and modernized technology stack that is engineered from the start to support a federated deployment topology with far less centralized administration.

Scope
This Test Plan prescribes the scope, approach, resources, and schedule of the testing activities. It identifies the items being tested, features to be tested, testing tasks to be performed, personnel responsible for each task, and risks associated with this plan.

The scope of testing on the project is limited to the requirements identified in the project’s Knowledge Repository Requirements Specification. The project has been broken up into four phases (Inception, Elaboration, Construction, and Transition) with one month iterations in each. Requirements for separate functional areas are determined at the start of each iteration. The impact of this on the Test Plan is that the specifics of the test plan and the test data will be determined as requirements are included in the SRS.

Resources

Team Members

<table>
<thead>
<tr>
<th>Role</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Director</td>
<td>Joshua Phillips</td>
</tr>
<tr>
<td>Project Manager</td>
<td>Beate Mahious</td>
</tr>
<tr>
<td>Subject Matter Expert</td>
<td>Ram Chilukuri</td>
</tr>
</tbody>
</table>
Software Test Strategy

Objectives

The Knowledge Repository will result in a production system that is fully functional with respect to the requirements. The overall object of this test plan is to provide unit, integration, and quality assurance testing for the whole of the delivered software. Unit testing is done during code development to verify correct function of source code modules, and to perform regression tests when code is refactored. Integration tests verify that the modules function together when combined in the production system. User acceptance testing verifies that software requirements and business value have been achieved.

Approach

The testing approach is to convert the use cases described in the use case document into a number of automated unit and integration tests to ensure the software conforms to the requirements. The following proposes the approach for testing the Knowledge Repository:

- Create a clear, complete set of test cases from the use case documents and review it with all stakeholders.
- Throughout the project, maintain the Requirements Traceability Matrix so that any stakeholders or tester has a concise record of what tests are run for each requirement.
- All test cases will be command line accessible to take advantage of continuous integration testing thru the use of ANT for all testing phases.

Some of the practices that the SemanticBits team will adopt are:
Load is a measurement of the usage of the system whereby a server is said to experience high load when its supported application is being heavily
workload. This can be measured in terms of response time (the time it takes between initial request and the response).

SemanticBits has in-depth experience developing and applying non-functional tests, such as performance, load, scalability, and stress testing. All these
Non-functional Testing
tiers.

deployed services and web applications. This will be especially important starting in iteration 7 where SemanticBits will begin deploying against the NCI

These test often run end-user workflows that can be automated on existing (sometimes production) systems. We can provide system tests for the
components, initialize them, test them, and tear them down. We will apply Integration Tests specifically to the services, where we will follow the pattern
Integration testing validates the integration of components within or across systems. These tests can and should be automated and require no external
code dependencies though may require non-code dependencies (such as a database). A typical Integration Test pattern is to automatically deploy
components, initialize them, test them, and tear them down. We will apply Integration Tests specifically to the services, where we will follow the pattern
above. For example, we will be able to in an automated way build, configure, deploy, and cross-test each of the MDR, Model, and Knowledge
Management services together. This type of integration testing is necessary to insure that these critical systems can work together appropriately.

System testing validates deployed systems. These tests can and should be automated, though they require that systems (code) be manually deployed. These test often run end-user workflows that can be automated on existing (sometimes production) systems. We can provide system tests for the
deployed services and web applications. This will be especially important starting in iteration 7 where SemanticBits will begin deploying against the NCI
tiers.

Non-functional Testing

SemanticBits has in-depth experience developing and applying non-functional tests, such as performance, load, scalability, and stress testing. All these
tests fall in the same category and are closely related to each other. Performance test touch on the aspect of a system's speed under a particular
workload. This can be measured in terms of response time (the time it takes between initial request and the response).

Load is a measurement of the usage of the system whereby a server is said to experience high load when its supported application is being heavily
 trafficked. Scalability tests whether an application will have a response time that increases linearly as load increases and will be able to process more and
more volume by adding more hardware resources in a linear (not exponential) fashion. This type of testing has two forms:

• Test response time with the increase in the size of the database
• Test response time with the increase in the number of concurrent users
The purpose of load and scalability testing is to ensure the application will have a good response time during peak usage. Stress testing is a subset of load testing that is used to determine the stability of a given system or entity. It involves testing beyond operational capacity, often to a breaking point, in order to observe the results. SemanticBits has extensive experience in using tools such as InfraRED for profiling and diagnosing problems associated with the non-functional aspects of the system. We have successfully used this tool for C3PR and caAERS, as well as understanding performance implications of the COPPA NCI Enterprise Services.

Description of Functionality

See the following documents:
- Requirements
- Use Cases

Dependencies and Assumptions

**Java Programming Language**: the Knowledge Repository is developed in the Java programming language. The Java 6 SDK is being used for development. Integration tests and other tools and utilities will be written in ruby, groovy, or other appropriate languages that are useful in the testing environment. These languages provide some features that are not available in Java.

**Application Server**: The C3PR implementation requires a Java application server. Apache Tomcat and the Globus container will be used for development and testing.

**Relational Database**: The backend database targets both Postgres and Oracle relational databases. Unit tests will be run against both target databases.

**Web Browser**: User acceptance testing and integration testing will target the Internet Explorer 6.x/7.x and Firefox 2.x web browsers.

General Criteria for Success

Criteria for overall success are 100% success of all automated unit tests and most tests are satisfactory successful of the manual tests. Focus in phase I will be on automated testing, and focus in phase II will be on manual user acceptance testing and performance testing.

Readiness Criteria

Tests will be ready to be written when the following criteria have been met:
- Use cases are complete
- Use cases are translated into executable tests
- APIs are available for individual modules

Tests will be ready to be run when:
- Source code for individual modules is available and runnable
- The tests are written
- Dependent services are deployed

Pass/Fail Criteria

The follow criteria will be employed for determining the success of individual tests:
- Appropriate data returned: equality comparison of results to locally cached data
- Performance: documentation of performance in time and subjective determination that performance is acceptable for the complexity of the operation

Completion Criteria

The criteria for completion of the testing procedures is that the system produces the output desired by the user within expected performance requirements. Testing is considered completed when:
- The assigned test scripts have been executed.
- Defects and discrepancies are documented, resolved, verified, or designated as future changes.

Acceptance Criteria

For user acceptance testing, a range of bug severities will be employed such that a severity can be assigned to the success of each test case. For example, a tester could assign acceptable, acceptable with issues, unacceptable. For unit, system, and integration testing, acceptance is determined by the automated testing completing successfully.

When testing is complete, the software is acceptable when the test manager and project manager determine that existing unresolved issues are documented and within subjective tolerance. Both user acceptance testing and automated system/integration/unit tests will be taken into consideration.
Software Test Environment - General Environment

Subsequent sections are to describe the software test environment at each intended test site.

The Test Environment: The Test Environment is a stable area for independent system and integration testing by the Test Team. This area consists of objects as they are completed by Developers and meet the requirements for promotion. This environment ensures that objects are tested with the latest stable version of other objects that may also be under development. The test environment is initially populated with the latest operational application and then updated with new changed objects from the development environment.

The Acceptance Testing Environment: The acceptance-testing environment provides a near-production environment for the client acceptance testing. The release is delivered by the SCM group and managed by the client.

Software Items

- Java 1.6.x: used to run the Java programs that make up the tests
- Ant 1.6.x: used to run automated tests in batch
- JUnit 3.x/.4.x: used to implemented specific stateless test cases for automated unit testing
- Microsoft Word: used to document testing activities
- Microsoft Excel: used to document testing activities
- SVN: used to version test results

Hardware and Firmware Items

- Continuous build machine: TBD
- Test deployment machine: TBD

Other Materials

None.

Participating Organizations

The testing group consists of the project’s Test Manager, and the Tester(s). The groups listed below are responsible for the respective types of testing:

- **Unit Testing**: Development team members from SemanticBits will be responsible for conducting the unit tests.
- **Integration Testing**: Development team members from SemanticBits will be responsible for conducting the integration tests.
- **User Acceptance Testing**: The QA team from NCI will perform User Acceptance Tests.

Test Schedules

The Test Manager will coordinate with the Project Manager and add the planned testing activities to the master project schedule. Refer to the project SDP and schedule for additional information.

Risks

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Date Surfaced</th>
<th>Status</th>
<th>Impact</th>
<th>Likelihood</th>
<th>Mitigation Strategy</th>
<th>Mitigation Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>