

1. caAERS_test_plan .....	2
1.1 Document Change History .....	2
1.2 Introduction .....	2
1.3 Software Test environment .....	4
1.4 Software Test Strategy .....	4
1.5 Test Schedules .....	9

# caAERS\_test\_plan

## *caAERS Application Development* Test Plan

**Submitted By:** SemanticBits, LLC  
**Date:** March 2008  
**Document Version:** 1.00

## Document Change History

Version Number	Date	Description
1.0	March 2008	Draft of Test Plan for caAERS v1.5
1.1	March 2011	Update team members

## Introduction

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.

## Scope

This document provides instruction and strategy for incorporating Software Testing practices and procedures into the Cancer Adverse Event Reporting System (caAERS) project. This document demonstrates the application of testing on this software. The purpose of this project is to develop and to deploy an adverse event reporting system that is nationally scalable with a robust architecture to meet the needs of the caBIG™ Community. The caAERS requirements, data model, and use cases developed during Release 2.0 will be the foundation for this effort. The Developer Team will execute Elaboration, Construction and Transition Phase activities for this project. The project will be carried out using the Agile Unified Process Framework, emphasizing continuous integration, testing, and risk management. Two Adopters, Wake Forest and The Mayo Clinic have been assigned and funded by caBIG™ and their input and collaboration throughout this project is critical to the timely release of functional software. The Developer Team will work closely with these Adopters, any additional adopters identified by NCICB, and the Adverse Events Special Interest Group to ensure the deliverables of this project will meet the needs of the caBIG™ Community.

The scope of testing on the project is limited to the requirements identified in the project's Software Requirements Specification (SRS). The project has been broken up into three phases (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.

## Identification

caAERS 2.0

## Document Overview

This Test Plan defines the strategies necessary to accomplish the testing activities associated with caAERS. We expect that it will be further developed as the development project proceeds. Testing procedural instructions are included in the Standard Operating Procedures (SOPs). The remaining Test Plan sections are organized as follows:

- **Section 3: Software Test Strategy:** Describes the overall approach and techniques to be used during testing.
- **Section 4. Software Test Environment:** Describes the intended testing environment.
- **Section 5. Test Schedules:** Contains or references the schedules for conducting testing.
- **Section 6. Risks:** Identifies the risks associated with the Test Plan.
- **Appendix A. Acronym List:** Defines the acronyms used on the project.

## Resources

The table below lists the members of the caAERS team.

Name	Title/Role
Ram Chilukuri	Project Director
Denis Krylov	Architect
Paul Baumgartner	Project Manager / Business Analyst
Wesley Wiggins	Business Analyst
Biju Joseph	Lead Developer
Srini Akkala	Lead Developer
Monish Dombla	Senior Software Engineer
Ion Olaru	Senior Software Engineer
Karthik Iyer	Software Engineer & QA Lead
Nikhil Pingili	Software Engineer
Ben Traynam	Usability and UI Engineer
Sharon Elcombe	Institutional Lead, Subject Matter Expert
Sonja Hamilton	Institutional Lead
Jennifer Frank	Subject Matter Expert
Jean Hanson	Subject Matter Expert
Robbin Peterson	Subject Matter Expert
Bradley Andersen	Technical
Michael Carston	Technical
Kimberly Johnson	Institutional Lead, Subject Matter Expert
Nimesh Patel	CALGB Project Manager
Ajay Patel	Technical
Debbie Sawyer	Subject Matter Expert

Kelly Coligan	Subject Matter Expert - Training
Robin Heinze	Subject Matter Expert - Statistics
Allison Booth	Subject Matter Expert
Bob Morrell	Institutional Lead, Subject Matter Expert
Steven Cheng	Technical
Del Jones	Subject Matter Expert

## Software Test environment

This section describes the software test environment at each intended test site.

### SemanticBits

*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.

There exists infrastructure labeled as the 'QA tier' that apportions resources for testing. This includes dedicated hardware and software resources to ensure isolation from the development environment. Based on input from the development team, the QA will build specific tags of the code and validate test scripts and bugs against that build.

*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 1.5.0\_07: Used to run the Java programs that make up the tests

Ant 1.7.0: Used to run automated tests in batch

Ivy 2.0.0: Used for dependency management and resolution.

JUnit 3.8.1: Used to implement specific set of interdependent test cases for automated unit testing. (Note: Each test has as much state as needed to run test, but the tests are not interdependent. That is, no test relies on the outcome of any other test.)

Easymock 2.2.1: Used for dynamic mock objects.

Dbunit 2.1: Used for loading test data and unit test.

Subversion (SVN): Used to version tests.

Hudson 1.1x: Continuous build and testing application. Used as continuous build application and for archiving test results. (Note: New versions of Hudson are released regularly (approximately once a week). We upgrade our version of Hudson periodically throughout project.

### Hardware and Firmware Items

Continuous build machine:

Hudson Builder [+http://tools.semanticbits.com/hudson/job/caaers-postgres/](http://tools.semanticbits.com/hudson/job/caaers-postgres/)

Test deployment machine:

Internal machine, not accessible outside firewall

Public Test Machine: <https://demo.semanticbits.com/caaers/public/login>

### 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 end-user representatives perform User Acceptance Testing, which includes Mayo Clinic, Wake Forest, and CALGB SMEs.

# Software Test Strategy

## Objectives

The overall object of this test plan is to provide unit, integration, and quality assurance testing for the whole of the caAERS 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 caAERS 2.0 use case document into a number of automated unit and integration tests to ensure the software conforms to the requirements. The following approach for testing caAERS 2.0 is proposed:

- Create a clear, complete set of test cases from the use case documents and review it with all stakeholders.
- 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:

- Derive test cases/unit tests from updated functional specifications of all relevant use cases. Unit tests and testing scenarios will be constructed in parallel with core development efforts, while validating the specifications against the relevant use cases. The use of diagrammatic representations of use cases in the form of task-based flow-charts, state diagrams, or UML sequence diagrams may facilitate creation of test cases and monitoring outcomes.
- Teaming testers with developers to provide close coordination, feedback, and understanding of specific modules for testing purposes.
- Ongoing peer-review of design and code as a team based form of software inspection. Each developer will review and run code written by another developer on a regular basis (acting as QA inspectors in turns), along with joint code review to gain consensus on best practices and common problems.
- Automated test execution using Ant and unit testing to support rapid testing, capturing issues earlier in the development lifecycle, and providing detailed logs of frequent test results (through nightly builds). The automated test environment will be carefully setup to ensure accurate and consistent testing outcomes.
- Regression testing ensures that the changes made in the current software do not affect the functionality of the existing software. Regression testing can be performed either by hand or by an automated process. Developers will run regression test on individual workstations when they fix a bug. This ensures quality control before code check in.
- Continuous tests continuously run regression tests, providing rapid feedback about test failures as source code is edited. It reduces the time and energy required to keep code well-tested, and prevents regression errors from persisting uncaught for long periods of time. Hudson is used to build and run tests in an automated fashion. In addition, Hudson provides notifications to concerned parties about build / test failures. This helps in timely responses to problems due to code check-ins. Potential problem are identified early on and fixed.
- Integration and System testing tests multiple software components that have each received prior and separate unit testing. Both the communication between components and APIs, as well as overall system-wide performance testing should be conducted. This is done in as a mix of automated and manual testing.
- Usability Testing to ensure that the overall user experience is intuitive, while all interfaces and features both appear consistent and function as expected. Comprehensive usability testing will be conducted with potential users (non-developers) with realistic scenarios and the results will be documented for all developers to review.
- Prioritization of feature implementation and bug fixes with feedback from testers using the risk management matrix.
- Bug-tracking and resolution will be managed by regularly posting all bugs and performance reports encountered in JIRA, with follow-up resolution and pending issues clearly indicated for all developers and QA testing personnel to review.

## Unit Testing

During system development, each developer performs unit testing of his or her code before it is finished. Unit testing is implemented against the smallest non trivial testable element (units) of the software and involves testing the internal structure, such as logic and data flow, and the unit's functional and observable behaviors. The centrepiece of the caAERS unit testing strategy will be the JUnit unit-testing framework and will be augmented by DBUnit. The Spring Framework's mock library and EasyMock will be used to create mock objects to assist in isolating unit tests. Examples of unit testing for caAERS 2.0 software are as follows:

- Design and develop the interface for a non-trivial Java class.
- Write test case using JUnit testing all methods in the interface.
- As the class is developed the test case is run to ensure the class is working properly

Hudson will run unit tests each time the repository is revised and will archive test reports. Hudson can be configured to run unit tests based on various milestones being achieved. It can be triggered to run nightly builds or everytime code is checked in. Test reports will be generated and reported to the Project manager, Developer and QA. Developers will be able to run unit tests as needed to verify correct function of their code locally as well, however the results of these ad-hoc unit tests will not be saved.

## Integration Testing

Integration testing is a form of "white-box testing" where this testing method makes sure that the correct outputs are produced when a set of

inputs are introduced to the software. In this case, the software internals are visible and closely examined by the tester. The business logic of the system, including grid services, will be tested. The set of unit tests will be implemented with Ant, JUnit, and performance profiling tools. The following sections describe how each of the components will be tested.

## Grid Services

Grid Services will be tested using the caGrid system testing infrastructure, which provides facilities to dynamically build, deploy, invoke, and tear down grid services. Business logic will be tested by providing contrived inputs to the services and comparing the outputs to known values.

## ESB

The Enterprise Service Bus will be tested using the JUnit framework. Messages will be sent to the ESB and appropriate routing and construction of output messages will be validated.

## User Acceptance Testing (UAT)

Acceptance Level Testing represents the final test action prior to deploying any version of caAERS. Adopters will perform Acceptance Level testing using one or more releases of caAERS to verify its readiness and usability as defined in the use-case(s) and supporting documents. Subject matter experts will test the end-user application (web interface) and determine its acceptability from a usability standpoint. Each use case and requirement will be translated into at least one test case. The focus of these test cases will be on final verification of the required business function and flow of the system, emulating real-world usage of the system. To facilitate the UAT, we plan to engage the subject matter experts throughout the software development lifecycle, especially during the use case collection and prototyping sessions. We also plan to provide access to the interim builds of the system to the subject matter experts so that they can gain familiarity and provide valuable feedback for increasing the usability of the system. The development team will closely work with subject matter experts during the UAT. In addition, SMEs will be provisioned with patches to a currently released version to facilitate continuity in testing.


User acceptance testing will be performed by the following individuals:

- Sonja Hamilton, Jennifer Frank, Jean Hanson: Domain Expert/Mayo CCC
- Bob Morrell, Steven Cheng, Kim Livengood: Domain Expert/WFU
- Kimberly Johnson, Susan Sutherland, Debbie Sawyer, Nimesh Patel, : Domain Expert

Please update if this info is old.

## Section 508 Compliance Testing

Testing will be conducted to ensure that the caAERS application is in compliance with the following Section 508 accessibility requirements:

- (a) A text equivalent for every non-text element shall be provided (e.g., via "alt", "longdesc", or in element content).
- (b) Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.
- (c) Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup.
- (d) Documents shall be organized so they are readable without requiring an associated style sheet.
- (e) Redundant text links shall be provided for each active region of a server-side image map.
- (f) Client-side image maps shall be provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.
- (g) Row and column headers shall be identified for data tables.
- (h) Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.
-  (i) Frames shall be titled with text that facilitates frame identification and navigation.
- (j) Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.
- (k) A text-only page, with equivalent information or functionality, shall be provided to make a web site comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.
- (l) When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.
- (m) When a web page requires that an applet, plug-in or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).
- (n) When electronic forms are designed to be completed on-line, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.
- (o) A method shall be provided that permits users to skip repetitive navigation links.
- (p) When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.

In addition, the functional test scripts that are created for UAT testing will include, where appropriate, steps to verify and test the accessibility features of the caAERS application.

A report will be created to show the results of the caAERS 508 compliance testing.

## HIPAA Compliance Testing

HIPAA, the Health Insurance Portability and Accountability Act, requires healthcare organizations to take added precautions to ensure the security of their networks and the privacy of patient data. As stated in the SRS document, caAERS is required to conform to applicable HIPAA specifications.

There are three broad classes of HIPAA requirements:

### HIPAA Privacy Rule

The HIPAA Privacy Rule mandates the protection and privacy of all health information. This rule specifically defines the authorized uses and

disclosures of "individually-identifiable" health information.

#### **HIPAA Transactions and Code Set Rule**

The HIPAA Transaction and Code Set Standard addresses the use of predefined transaction standards and code sets for communications and transactions in the health-care industry.

#### **HIPAA Security Rule**

The HIPAA Security Rule mandates the security of electronic medical records. Unlike the Privacy Rule, which provides broader protection for all formats that health information make take, such as print or electronic information, the Security Rule addresses the technical aspects of protecting electronic health information. More specifically, the HIPPA Security standards addresses these aspects of security:

- Administrative security\* – assignment of security responsibility to an individual.
- Physical security\* - required to protect electronic systems, equipment and data.
- Technical security\* - authentication & encryption used to control access to data.

While the caAERS application will be designed to support all the applicable HIPAA requirements, the burden for ensuring that the use of caAERS is in compliance resides with the adopting institution. That is, it is the responsibility of the institution to ensure that the necessary business, technical, and security policies and procedures are put into effect and enforced and that the necessary physical security safeguards are in place.

Our testing will be designed to ensure that the technical security requirements are satisfied by the caAERS application. Test scripts will be designed to test each of the HIPAA Security Rule items marked for testing in the table below.

<b>Description. R=Required. A=Addressable</b>	<b>Testing Strategy</b>
<b>Access Control (R)</b> – Include mechanism to allow access only to those persons or software programs that are authorized.	To be tested.
<b>Unique User Identification (R)</b> – Assign a unique name and/or number for tracking user identity.	To be tested.
<b>Emergency Access Procedure (R)</b> – Establish procedures for obtaining necessary electronic protected health information during an emergency.	No specific testing. Because caAERS is a web based application accessible by browser, an Administrator should be able to access the application and data from anywhere at anytime within SSL.
<b>Automatic Logoff (A)</b> – Include mechanism that terminates an electronic session after a predetermined time of inactivity.	This is configurable in caAERS Admin page and has been incorporated as part of normal test scripts
<b>Encryption and Decryption (A)</b> – Include mechanism to encrypt and decrypt electronic protected health information.	This is ensured by use of SSL and public key certificates.
<b>Audit Controls (R)</b> – Include mechanism that records and examines activity in information systems that contain or use electronically protected health information.	Audit controls exist which record identity of users and time stamp when data was modified.
<b>Integrity (R)</b> – Implement mechanism to protect electronic protected health information from improper alteration or destruction.	This is implemented in caAERS through appropriate access controls. Users are assigned roles in caAERS which carry certain privileges with them. Testing for role based access to ensure integrity has been extensive in caAERS.
<b>Authenticate Electronic Protected Health Information (A)</b> – Include mechanism to corroborate that electronic protected information has not been altered or destroyed in an unauthorized manner.	This is implemented in caAERS through appropriate access controls. Users are assigned roles in caAERS which carry certain privileges with them. Testing for role based access to ensure integrity has been extensive in caAERS.
<b>Person or Entity Authentication (R)</b> – Include mechanism to verify that person or entity seeking access to electronic protected health information is the one claimed.	caAERS ensures authentication by password controls. The password policy implemented by caAERS allows for protection of user identity and correct authentication.
<b>Transmission Security (R)</b> – Include mechanism to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.	caAERS runs in the tomcat container over an SSL encrypted connection to forbid un-authorized eavesdropping of sensitive information.
<b>Integrity Controls (A)</b> – Include mechanism to ensure electronically transmitted electronic protected health information is not improperly modified without detection until disposed of.	To be tested.
<b>Encryption (A)</b> – Include mechanism to encrypt electronic protected health information whenever deemed appropriate.	To be tested.

## Description of Functionality

See the following documents.

Use Case Document:

[https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/docs/Use%20Cases/caAERS\\_draft\\_Use\\_Case.doc?root=caaersappdev&view=log](https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/docs/Use%20Cases/caAERS_draft_Use_Case.doc?root=caaersappdev&view=log)Update this link

SRS Document:

[https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/docs/SRS/caAERS\\_draft\\_SRS.doc?root=caaersappdev&view=log](https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/docs/SRS/caAERS_draft_SRS.doc?root=caaersappdev&view=log)Please update this link

## Specific Exclusions

NA

## Dependencies & Assumptions

### Java programming language

caAERS is developed in the Java programming language. The Java 5 SDK is being used for development. Integration tests and other tools and utilities are written in Java due to the availability of mature enterprise grade tools

### Application Server

The caAERS 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 PostgreSQL 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 7 and Firefox 2+ web browser.

## General Criteria for Success

Criteria for overall success are 100% success of all automated unit tests and most tests are satisfactorily successful of the manual tests. Focus in phase I Update phase here 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 are 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. Any deviations of the system from expected behavior is recorded as a bug in JIRA. They are in turn cross-referenced in the test script results to facilitate tracking.



## 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 test 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.

## Test Schedules

### Time Frames for Testing

The Test Manager will coordinate with the Project Manager and add the planned testing activities to the master project schedule. Reporting bugs and verifying bug fixes are part of the ongoing project schedule. In addition, the Test manager will report resource requirements to the project manager for planning the iteration. Project Manager will schedule resources for running smoke tests prior to milestone releases. Unit and Integration Testing will be performed through the lifetime of the project.