CTMS : C3PR Test Plan

This page last changed on May 22, 2008 by McconneP.

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 C3PR project.

The Cancer Central Participant Registry (C3PR) is a web-based application used for end-to-end registration of patients to clinical trials. This includes capturing the consent signed date, eligibility criteria, stratification, randomization, and screening. Clinical workflows are enabled by both subject- and study-centric views into the registration process. C3PR can be run in a standalone mode where study definitions, investigators, study personnel, and sites are entered into the system, or C3PR can be run in an integrated mode with the caBIG Clinical Trials Suite (CCTS). C3PR also enables multi-site clinical trials where registration information is entered locally at affiliate sites and the registration is completed by call-out to the coordinating site.

Throughout the development of C3PR, a number of elaborator and adopter sites are actively being engaged to help define requirements and test the application. Our primary elaborators include Duke, Wake Forest, Mayo, Westat, CALGB, CCR, and the Coalition of Cooperative Groups. Our primary adopters include Duke and Wake Forest with engagement of Georgetown and CCR.

C3PR release 1 was developed by Nortel Solutions and released in 2006. Release 2 was developed by Duke Cancer Center in collaboration with SemanticBits, LLC and was released in March, 2008. We are currently in the next phase of development with releases slated for the end of September, 2008 and March, 2009.

See our Vision Statement for more information.

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

C3PRv3

Document Overview

This Test Plan defines the strategies necessary to accomplish the testing activities associated with C3PR. 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 Test Plan sections are organized as follows:

Resources

Team	Members
Duke	• Jamie Cuticchia (PI) • Bob Annechiarico (Project Director, Co-investigator) • Pankaj Agarwal (Project Manager) • Mohammad Farid (DBA) • Peter Le (IT Analyst) • Vijaya Chadaram, RN (Subject Matter Expert) • Emily Allred (Admin)
SemanticBits	• Ram Chilukuri (Technical Director, Architect) • Patrick McConnell (Architect) • Kruttik Aggarwal (Lead Developer) • Ramakrishna Gundala (Developer) • Vinay Gangoli (Developer) • Himanshu Gupta (Developer) • TBD (Business Analyst) • Shilpa Alluru (QA Tester) • TBD (Technical Writer)
Wake Forest	• Bob Morrell (Institutional Lead, Subject Matter Expert) • Steven Cheng (Technical Tester) • Kim Livengood (Subject Matter Expert)
Mayo Clinic	• Sharon Elcombe (Institutional Lead, Subject Matter Expert)
CALGB	• Kimberly Johnson (Institutional Lead, Subject Matter Expert) • Amish Shah (IT Analyst)
Westat	• Steve Riorden (Institutional Lead, Subject Matter Expert) • etc.

Other Documents

Management	End User	Analysis	Technical
• Vision Statement • Project Plan • Scrum Artifacts • Adoption Plan	• End User Guide • Training Module • Installation Guide	• Use Case Document • Requirements Specification	• Architecture Guide • Domain Analysis Model

• Communications Plan • Test Plan	• Administration Guide	• Activity Diagrams	• Domain Implementation Model • Deployment Diagrams

Software Test Strategy

Objectives

C3PR 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 C3PR 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 C3PR 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 C3PR:

- 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 Duke 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. The regression testing will be achieved by using a nightly build.
- Continuous Testing uses excess cycles on a developer's workstation to continuously run regression tests in the background, 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
- 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.
- 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.

- Bug-tracking and resolution will be managed by regularly posting all bugs and performance reports encountered in GForge, 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 C3PR unit testing strategy will be the JUnit unit-testing framework and will be augmented by HTTPUnit, Schemamule, and DBUnit. Examples of unit testing for C3PR 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
  Complex test cases will be implemented with the Haste unit-testing framework. This will allow the gluing of test cases into a "story" that represents the overall unit being tested.

Hudson will run unit tests each time the repository is revised and will archive test reports. Developers will be able to run unit tests as needed to verify correct function of their code, 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, Haste, 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 Haste framework. Messages will be sent to the ESB and appropriate routing and construction of output messages will be validated.

GUI

The web interface will be tested using HTTPUnit, which mimics the functionality of a web browser. User scenarios will be constructed, data input into web forms, and the output validated.

User Acceptance Testing (UAT)

Acceptance Level Testing represents the final test action prior to deploying any version of C3PR. Adopters will perform Acceptance Level testing using one or more releases of C3PR 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 lead architect (Patrick McConnell) and lead developer (Kruttik Agarwal) will closely work with subject matter experts during the UAT.

Description of functionality

See the Use Case Document and the SRS

Specific Exclusions

Not Applicable

Dependencies and Assumptions

Java programming language

C3PR is developed in the Java programming language. The Java 5 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 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.

Software Test Environment

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

General Environment

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 1.5.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 end-user representatives perform User Acceptance Testing, which includes Duke and Wake Forest SMEs.

Test Schedules

Timeframes for Testing

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.

Unit and Integration Testing will be performed through the lifetime of the project. Quality Assurance testing will begin in month 3 of the elaboration phase.

Risks

Number	Risk	Date Surfaced	Status	Impact	Likelihood	Mitigation Strategy	Mitigation Description