Open Source Electronic Health Record (EHR) Refactoring Services

Master Test Plan

Version 1.0

March 2012

Revision History

Version	Description	Author	Reviewer(s)	Review Type	Issue Date
0.1	Initial Draft	E. Null	K. Keating, K. Cirka	Peer	02/10/12
0.2	Minor revisions	K. Keating, K. Cirka			
0.3	Minor revisions	K. Keating	E. Null	Peer	02/27/12
0.4	Address review comments/minor revisions	E. Null	K. Keating	Formal	03/02/12

Table of Contents

1 Introduction

The Open Source Electronic Health Record (EHR) Refactoring Services project will reorganize the Open Source EHR application Mumps (M) codebase (formerly known as the Freedom of Information Act, FOIA, instance of Veterans Health Information Systems and Technology Architecture, VistA) into modular components. The project will also facilitate the development of open standard interfaces that makes component functions accessible to modern programming practices to support three-tier architecture. Per Customer direction, the refactoring project has a limited scope of only refactoring at the application layer.

1.1 Purpose

The purpose of this Master Test Plan (MTP) is to define the procedures and activities used to validate that the refactoring solution meets its defined requirements by:

- Providing a central document to control the test effort; it defines the approaches that will be employed to test the software and to evaluate the results of that testing, and is the top-level and detailed plan that will be used by managers to govern and direct the testing work
- Providing visibility to stakeholders in the testing effort that adequate consideration has been given to various aspects of governing the testing effort, and where appropriate to have those stakeholders approve the plan

This plan details the following in accordance with the abovementioned requirements: the items that should be targeted by tests; the motivation for and ideas behind the test areas to be covered; the testing approaches that will be used; and system test processes and procedures for pre-test preparation, test readiness reviews, exit criteria, and reporting. It also describes the roles and responsibilities of those involved in the system test process.

1.2 Scope and Audience

This scope of this document is limited to the testing at the application layer of refactored components. The MTP will be updated as necessary as additional applications/components (henceforth referred to as *modules*) are included in the refactoring project's efforts.

The processes outlined in this document are intended for the project team, subject matter experts (SME)s, and stakeholders who are working on testing activities, or those who need to be aware of the information it contains. This document and its contents provide guidance for Testers at every level of testing.

Stakeholders should be aware that this MTP references approved testing procedures as governed by the National Institute of Standards and Technology (NIST) Standards and Testing. Additional testing resources are cited in Section 1.6.

1.3 Testing Objectives

This MTP supports the following test objectives:

- Adaptation to development methodology, the goal being to test for proper functionality as early as possible
- Identification and configuration of the test environment provided by the Open Source Electronic Health Record Agent (OSEHRA)

- Creation, maintenance, and control of the Test environment
- Providing test coverage for 100% of the documented requirements for VA and OSEHRA
- Execution of 100% of the initial test cases during functionality testing and integration testing (documentation and data provided by VA via OSEHRA)
- Ensuring the software interfaces correctly with existing systems
- Guaranteeing that the application meets or exceeds performance expectations and capabilities for OSEHRA
- Verification of Section 508, Security, and Privacy compliance
- Demonstration that usability standards are met

1.4 Development and Testing Approach

Modules selected as refactoring candidates are refactored through the course of one or more Sprints. During the refactoring process, testing approaches are defined and specified for the module being refactored based on different dependencies the chosen module may have with others within the entire Open Source EHR (FOIA VistA) codebase. The testing approaches for each module are detailed in an Appendix to this MTP.

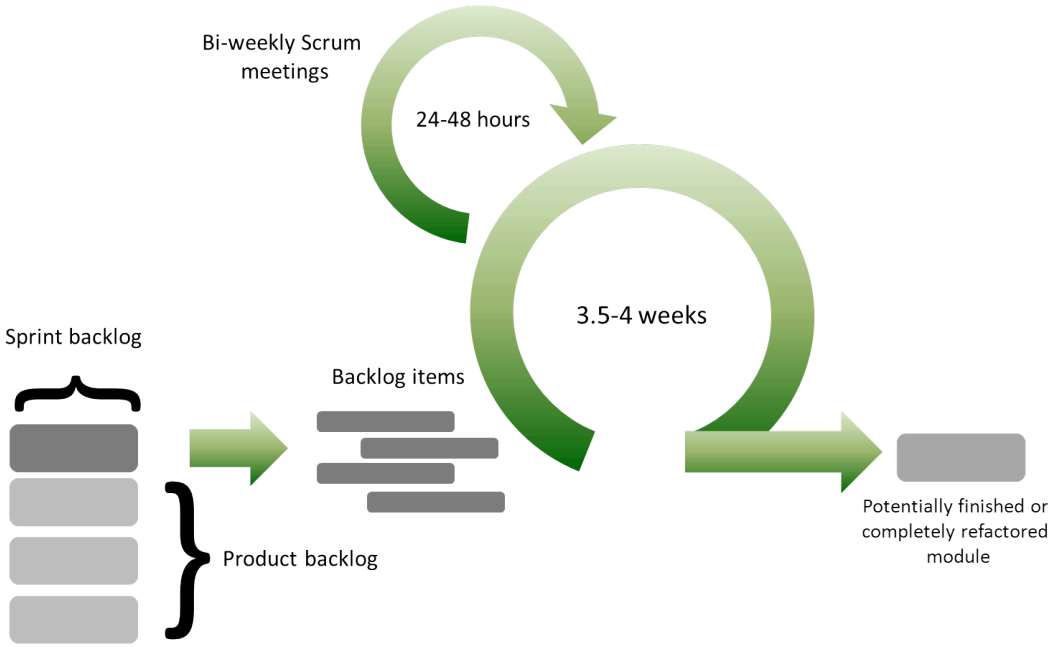

Figure 1: Enterprise Context Diagram

1.5 Roles and Responsibilities

The following table lists the key roles and responsibilities for this MTP:

Table 1: Testing Roles and Responsibilities

Role	Responsibilities
Test Lead	Individual that and coordinates activities related to all aspects of testing based on an approved Master Test Plan and schedule ensuring; ensures full execution of the test process to include the

Role	Responsibilities
	verification of technical requirements and the validation of business requirements, as well as making sure the test environment will adequately supports planned test activities
Chief Architect	Individual responsible for leading and coordinating activities related to all aspects of building or construction of the product; also assists with the creation and implementation of the MTP
Development Team	Persons that build or construct the product/product component
Systems Architect	Individual that assists in testing creation and execution
Stakeholders	Individual(s) that hold a stake in a situation in which they may affect or be affected by the outcome; these individuals will assist in testing

1.6 References

Information in the following documents supplements the information in this plan:

- http://healthcare.nist.gov/use_testing/effective_requirements.html Criteria 170.302(c) Maintain up-to-date Problem List. http://healthcare.nist.gov/docs/170.302.c_problemlist_v1.1.pdf
- Computerized Patient Record System (CPRS) Problem List User Manual ,December 2004, Health Data Systems Veteran Health Administration, www.va.gov/vdl/documents/Clinical/CPRS-Problem_List/gmplum.doc

1.7 Acronyms and Definitions

The following table defines acronyms used in this document.

Table 2: Acronyms and Definitions

Acronyms	Definition
API	Application Programming Interface
CPRS	Computerized Patient Record Systems
DB	Database
EHR	Electronic Health Record
GUI	Graphical User Interface
KIDS	Kernel Installation and Distribution System
M	Mumps (code)
MTP	Master Test Plan
NIST	National Institute of Standards and Technology
RPC	Remote Procedure Call
SME	Subject Matter Expert
TDD	Test-Driven Development
VistA	Veterans Health Information Systems and Technology Architecture

1.8 Processes

The processes that guide the implementation of this testing plan are:

- Test preparation
- Build process
- Test execution
- Functional Testing
- Defect Management
- Test approval
- Section 508 Compliance Testing
- Performance Testing
- OSEHRA Certification and Accreditation

1.8.1 Test Preparation

Testing is performed to validate the development (refactoring) work. Test planning begins as soon as builds are deployed in to the Development and Testing environments, and continues through the end of the release. Specific to this project, a 'release' refers to the submission of a completed refactored module to OSEHRA for certification.

1.8.2 Build Process

Once the module(s) to be refactored is/are identified, our team will follow Scrum, an Agile methodology, to deliver software; Sprint cycles will be three to four weeks long. Developers will use various tools such as XINDEX utility, read documentation and the codebase, and consult with OSEHRA resources and external collaborators to understand various dependencies and functionality. For actual development, Intersystems Cache Studio (henceforth referred to as 'Cache') and Eclipse will both be utilized as Integrated Development Environments to deliver results; these results will be applicable in writing brief developer-level unit tests. In addition, MUnit will be used for Application Programming Interface (API) unit testing.

The refactored code will be housed in public github repository, *kthlnkeating/VistA-FOIA*, at *http://www.github.com*. This refactored code will initially be uploaded to testing environments using tools provided by OSEHRA and detailed in *http://osehra.org/wiki/instructions-establishing-and-testing-osehra-code-base-main-page*. The utilized testing environment can either be local Cache or GTM installations on Testers' local machines or RGI's Cache installation on *vista.raygroupintl.com*. In the future, Kernel Installation and Distribution System (KIDS) packages will be created, specific to each build (the final refactored code). Before final code release, all tests will be repeated on a testing environment that is updated using KIDS packages.

1.8.3 Test Execution

Multiple levels of testing are performed to address different areas of refactoring and structural changes. Developers are responsible for conducting unit testing for their code before it is checked in to the Test environment. After the code is checked in a build is completed.

Builds from the Development environment are promoted to the Test environment after a successful completion of unit testing. Testers conduct formal testing using detailed test cases and test scripts. For each Sprint, the Testers conduct testing of the newly refactored modules.

Test execution records show the test execution activities. In order for the test case and associated scripts to pass all of the execution, records must show that the test executions have a 'Pass' indicator with no critical defects. For this project, working test execution records will be kept in an Excel spreadsheet and updated each time a test is performed. Records will illustrate the following:

- Type of test performed
- Date the test was performed
- Test environment
- Test case ID or test script ID
- 'Pass' or 'Fail' indicator
- Tester comments
- Subsequent retest dates and the indicator for each test
- Defect numbers for each defect found for the test execution (linked to the issue and defect logs)

When test cases are approved and completed with all test criteria successfully met, then the test case is completed. After the refactored module passes these internal project team testing procedures, it may be released to OSEHRA for acceptance and certification. Defects may be entered against a test case if functional issues are found within the refactored module.

1.8.4 Functional Testing

Functional testing is based upon the *black box* technique, defined as verifying the application internal processes by interacting with the application via the Graphical User Interface (GUI) and analyzing the output or results. This testing verifies functional, performance, and reliability requirements placed on major design items. These "design items" are assemblages (groups of units) that are exercised through their interfaces. Test cases are constructed to test that all modules within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules (previously referred to as unit testing). The overall idea is to use a building block approach in which verified assemblages are added to a verified base, which is then used to support the integration testing of further assemblages.

In addition to using a CPRS GUI based test approach, Cache may be used to connect using a Terminal. Non-GUI based functional testing may be done using VistA's scroll-and-roll interface; the scroll-and-roll interface requires the Tester to have an intimate knowledge of Cache and M.

Testers using a GUI, such as CPRS, can test at a more functional level. It is recommended that when possible the refactored modules of the VistA application layer be tested using both, a GUI interface (CPRS) and the scroll-and-roll Terminal.

Test case and test script development will be completed by the testing team using a varied mix of resources including NIST, existing user documentation for VistA, and analysis of the existing systems. To the extent possible, any existing test scripts that can be leveraged will be incorporated in to the test plan for completeness.

1.8.5 Defect Management

A defect has been defined as '*any unwanted behavior by the application*.' There are two classes of defects tracked in this refactoring project: those found before the code is released to OSEHRA for certification, and defects found during the OSEHRA certification process. This testing strategy scope is limited to testing that occurs before code is submitted to OSEHRA.

Internally, defects are identified by Testers during test execution; testing can initiate and terminate at different times throughout a Sprint. All defects found are logged using an Excel spreadsheet and will be linked to the project's Product Backlog for complete tracking purposes.

Defects can be identified when the following conditions occur:

- Actual results do not correspond to expected results described in the test cases
- Processing events do not correspond to what is described in the user stories and the requirements
- Other defects are found, such as field mapping and validation, incorrect error conditions, and usability issues
- The test case is unable to be repeated

1.8.5.1 Testing and Defects

When a developer confirms that their code is complete for a feature and the code is promoted to the Test environment through a build deployment, the test engineers begin formal testing on the code. When the test result of a test case deviates from the expected results, the actual results are documented.

Test results are recorded in the test execution records, and results that are not expected may identify issues and defects. Before determining if a test result is validated as a defect or an issue, the results must be verified with the developers. Most often a demonstration, executing the test scripts to show the unexpected result, is necessary.

Issue logs are reviewed with the developers before a formal defect is entered. The issue is analyzed and if it is determined to be out of scope, it is noted in the Product Backlog to be addressed during the Sprint Review for a future discussion. If valid, the issue is placed on the Product Backlog and either assigned to the current Sprint (i.e. included in an existing story for the current Sprint) or scoped for a future sprint during a Sprint Planning session.

The test cases are then reviewed to determine if the test is valid. For valid issues the developer may modify the code and attempt to fix the issue before the end of the Sprint. If the issue is fixed before the end of the Sprint and the test engineer verified the fix during the end of Sprint testing, then the issue is resolved with no defect being entered.

If issues are not resolved by the end of the Sprint, then defects are logged and added to the Product Backlog. Logged defects are prioritized and assigned to future Sprints or releases. Defects are resolved as developers are assigned to fix them and the fixes are verified by Testers; defect fixes can occur in any Sprint.

Defects are assigned a severity level depending on the type of defect and how it will impact the user's ability to use the system or complete work. Severity levels from 1 to 2 are serious defects and levels 3 and 4 are minor to moderate inconveniences which do not impact the user's ability to perform work or operate the application. For more information on defect severity levels, reference Section 3.5.

1.8.5.2 Recording Defects

When logging a defect, the following fields should be completed:

Summary	Completed by the person logging the defect and may be modified as needed. The summary should tell the reader which Epic or User Story and requirement the defect is entered against. An example could be: An example is: "Problem List – ICD 9 Code

	Behavior"
Severity	Field may be completed by the person creating the defect, but may change when triage occurs. This field corresponds to how the defect will affect the system overall based on the number it has been assigned; the person logging the defect should take into account how much of an impact will this defect has on the whole system if not fixed.
Filed Against	Field may be completed by the person logging the defect, but may change when triage occurs. Most defects will be initially filed against module development until they are triaged and then assigned to a specific developer on the project's Technical Team.
Owned By	Field may be completed by the person logging the defect, but may change when triage occurs. This field is set after the *Filed Against* field is set. This is set first to the Developer who is applying the change to fix the defect and then to the Tester who will verify the change.
Priority	Field may be completed by the person logging the defect but may change when triage occurs. Priority can have an Unassigned state or be set to the following: 1 (Resolve Immediately), 2 (Give High Attention), 3 (Normal Queue), and 4 (Low).
Planned For	Field may be completed by the person logging the defect, but may change when during Sprint Planning.
Estimate	Completed by the Developer assigned to work on the defect and the Tester who will validate that the defect is fixed. This is the estimated total number of hours that are estimated to complete the defect.
Correction	Completed by the Developer assigned to work on the defect and the Tester who will validate that the defect is fixed. This is the actual total number of hours that are spent to completely fix the defect.
Time Remaining	Completed by the Developer assigned to work on the defect and the Tester who will validate that the defect is fixed. The number of hours left from the estimated number of hours to complete the defect work.
Description	When completing the description field, every defect must have the following ten items listed. If one or more of the items in the list do not apply, list the number followed by 'N/A'. 1. The internet browser being used, display type/settings, and values 2. The environment where testing is being performed 3. Build number 4. User name and password of the test account used; note the roles and permissions for the user 5. Date and time (an approximate estimation will suffice) so the logs can be checked 6. The 'requirement' that is not being followed by the software; if the requirement is not known, then the User Story would be sufficient; the appropriate Scrum team may link the defect to the appropriate requirement for traceability 7. What are the steps to recreate the defect 8. Any screen shots and also data if it is back end and also test data (user

	credentials/file numbers); list the titles of the documents linked to the defect 9. Any queries used, name of the database validating against, and the table/field names 10. The actual results obtained which revealed the defect and why those results show a flaw, failure, or fault in the code.
Links	Any screen shots or other attachments that will help the developer and Tester resolve the defect may be added here. Add Related Epics or User Stories to the defect but do not assign them as a Parent or Child. Add Tested by Test Case to show which Test Case will test case will validate the defect.
Approvals	Set the Approvers who will validate that the test case can be closed after testing is completed and the defect is resolved. A defect is treated similarly to a User Story and requires the same level of approval.

1.8.6 Test Approval

For testing to be completed, the following criteria must be met:

- All related test cases have been executed
- Review of test results has been completed and approved
- Test cases 'Pass' or "Pass with constraint'
- Approval to exit the test may not be given if any test case fails with Severity 1 or 2 defects

1.8.7 Section 508 Compliance Testing

Not applicable for this project.

1.8.8 Performance Testing

Not applicable for this project.

1.8.9 OSEHRA Acceptance and Certification

After the refactored code has been successfully tested, it will be released to OSEHRA for certification. The following link will give the reader additional details on what the certification process entails: http://www.osehra.org/page/software-quality-certification/.

1.9 Items to be Tested

The modules and their respective features, and combinations of modules (or packages) and features that will be tested are:

- Problem List

As additional modules are included in the refactoring project's scope, they will be included as part of the MTP.

1.10 Overview of Test Exclusions

- Modules that have not been refactored
- CPRS or other EHR GUI systems used to test the refactored modules

2 Test Techniques

2.1 Testing Approach

2.1.1 System Tests

Developers are using the test-driven development (TDD) methodology where detailed and complete unit testing is done as development is completed.

2.1.2 User Functionality Test

All refactored code is tested as it is delivered for each Sprint. The overall objective of functional testing is to ensure that the refactored code did not have adverse effects on any dependent modules or other components of VistA. End-to-end testing is performed to validate:

- That the refactored code did not change the functionality implemented in the current version of VistA (both scroll-and-roll and CPRS)
- Interdependencies between the refactored module and other application modules, as well as those between scroll-and-roll and CPRS
- That the refactored code delivered in the current Sprint did not cause functional issues with refactored code previously delivered

2.1.3 Independent Testing

Please reference the OSEHRA Code Review site at the following address:
http://review.code.osehra.org/#/q/status:open,n,z

2.2 Testing Techniques

Table 3: Testing Types

Test Type	Technical Team Member Responsible
Build verification testing	Afsin Ustundag
Installation testing	Paul Bradley
Integration testing	Afsin Ustundag
Smoke testing	Afsin Ustundag
Usability testing	Ed Null/ Paul Bradley
User functionality testing	Ed Null/ Paul Bradley
User interface testing	Ed Null

3 Test Criteria

3.1 Entry Criteria

Preparing test cases and test scripts is required for the refactoring project's testing strategy. The test cases and scripts may be provided by other sources such as NIST or existing user documentation for the modules. To start system testing for each module, a test readiness review is conducted to verify that all entry criteria are met to ensure that a proper test environment is in place to support the system test process. Entry criteria to the system test case construction and execution are:

- All test hardware platforms must have been successfully installed, configured, and must be functioning properly
- All standard software tools, including testing tools, must have been successfully installed and must be functioning properly
- All documentation and design of the architecture must be available
- All personnel involved in the system test effort must be trained in the tools to be used during testing
- Proper test data is available
- Development is completed for testing
- Successful creation of build and deployment in test environment
- Unit test is successful and the results have been reviewed by the Test Lead
- Test cases and test scripts are peer reviewed
- System test data is available to complete the tests for the module to test
- Test readiness review is completed to verify all the above entrance criteria are met

3.2 Process Reviews

All functional/system test cases will undergo peer reviews. During peer review, each system test case is reviewed in detail to ensure that the test steps, input conditions, output, and validation points are correct to the technical refactoring requirements that are documented by the Technical Team. The members required to complete a peer review are:

- SMEs
- Test Lead
- Developer (optional)

If a test case should not pass review, the Tester responsible will update the test case to reflect the changes noted during the review. The test case will once again be reviewed with business analyst to ensure that the modifications have been made correctly.

3.3 Executing the Test

Each Tester will execute the system test cases they have created and collect test results as they execute each test case. System test cases may be executed in sets that have specific order of execution as denoted in the test case descriptions.

If a system test case fails at any point during execution of a test set that has a defined order of execution, the sequence must be stopped and the process for documenting defects must be completed (reference Section 1.8.5). In this situation, a Tester may proceed with those system test cases that have no such dependency.

Execution of system test cases and system test sets may be repeated multiple times as a direct or indirect result of changes being made to the system. When the exit criterion for the completion of the system test has been met for all system test cases under test, test execution may stop for that subsystem.

Additionally, test cases will be re-executed as part of regression testing at the system test level.

3.4 Collecting Test Results and Validating the Test

The Tester will be responsible for validating each test executed. As each test is executed, the Tester will collect the results of the test in the test execution record document. Depending on the nature of the test, results may be collected to demonstrate pre-conditions of the test, conditions during the test at each step of the test, and conditions after the test has been completed. These results are then compared to the expected results as recorded in each test case.

When the result is compared to the expected result at each validation point within the system test case, an assessment of the result is made in terms of pass or fail. Additionally, when the system test case is executed to completion, or as far as the subsystem allows, a 'Pass'/'Fail' determination is made for the entire test case. Therefore, it is possible for a system test case to be completed with minor issues detected at one or more points of validation and obtain an overall assessment of 'Pass'.

Testers will follow the defect tracking process should the results of the test validation indicate that a correction in the code, test case, or requirement is required.

Once all system test cases have met the acceptance criteria for exiting system test, a final assembly of all test results will be created to demonstrate to SMEs that the test has been completed successfully. This will include the test cases themselves, examples of database before and after images, screen prints, database queries, input files, processing records, and so on. Test results will be clearly labeled to facilitate the review process; this includes identifying each system test case and its corresponding results.

Within each test case result, specific validation points will be highlighted to allow easy comparison of expected and actual system test results. These results will then be reviewed by the development team and Testers.

3.5 Pass/Fail Criteria

Sometimes a 'Pass' or 'Fail' condition cannot be readily determined. The following guidelines illustrated in Table 5 are provided to assist all parties involved in testing:

- The severity level of a defect is assigned to show the importance of the defect and what kind of impact it may have
- A priority level is also assigned to rank the order with which the defects are addressed.

There are four priority levels: 1 (Resolve Immediately), 2 (Give High Attention), 3 (Normal Queue), and 4 (Low). For example, Technical Team members may have three severity 1 defects but two may have a priority of 1 and the other set to a 2. Severity 1/priority 1 defects are worked first, followed by priority 2, 3, and 4. Assessments of 'Pass' or 'Fail' criteria are made for each test case and test script.

Table 4: Pass/Fail Guidelines

Condition	Guideline
Pass (P)	The results of the action agree with what is expected
Pass with constraint	The results do not completely agree with expected results, but the results are just a slight inconvenience to the user (for example not the preferred number of results or mislabeled headers); a severity 3 or4 defect has been entered
Fail (F)	A) The function has failed, but there is a workaround such that testing of that subsystem can continue; a severity 1 defect has been entered B) The function has failed, and there is no workaround; a severity 2 defect has been entered C) The action results in a system crash or results in damage to data (for example, incorrect results or the loss of data); a severity 1 defect has been entered

Severity 1 issues are addressed in the same Sprint and severity 2 issues may be addressed in that Sprint or marked for fixes in a future Sprint; defects which are not critical and do not block other functionalities are addressed in future Sprints.

3.6 Reviewing the Test Results

Test results which are reviewed contain subsystem test cases and the corresponding results which were collected during execution. Each system test result will receive two types of reviews; there may be multiple reviews of each type. The types of reviews are:

- Technical Team Review
- SME Review

3.7 Suspension and Resumption Criteria

Suspension criteria are the criteria used to (temporarily) stop all or a portion of the testing activities on the test items. Resumption criteria are the testing activities that must be repeated when testing is re-started after a suspension. Testing may be suspended for the following reasons:

- Hardware/software not available at the time indicated during the planned project schedule
- The build contains many serious defects which seriously prevent or limit testing progress
- Assigned test resources are not available when needed by the Test Team
- Test environment is unavailable or cannot be efficiently utilized due to performance or other issues
- Ad-hoc testing failure
- Test data is unavailable
- Test data is invalid and causes module failure
- Insufficient knowledge transfer and no clarification for queries
- Unscheduled outage of the Testing environment

3.8 Resumption Criteria

If testing is suspended, resumption will only occur when the problem(s) that caused the suspension has/have been resolved. When a critical defect is the cause of the suspension, the new build must be verified by the Test Team.

3.9 Exit Criteria

Certain criteria must be met when testing is completed. The exit criteria for testing are:

- Refactored module must provide all of the required services relevant to the user specification
- All module documentation should be completed and up to date
- All critical defects (severities 1 and 2) should be verified
- All test cases that have no dependency on any other subsystem have been executed
- Test review of test summary and results has been completed and approved
- Test cases with no dependency on any other module pass or pass with constraint
- The appropriate sponsors (OSEHRA) have signed off indicating their agreement on the thoroughness and completeness of the testing and/or items to be differed or cancelled, if applicable

Approval to exit the module test may not be given if any test case that has no dependency on any other subsystem fails with severity 1 or 2.

Only the Project Manager can close testing if one or more test cases have failed with severity 3 level defects opened against them and all remaining test cases meet the criteria for exit as stated above.

3.10 Acceptance Criteria

The chosen candidate application passes internal testing and is passed to the OSHERA Certification Group for validation.

4 Test Environments

A test environment is an environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.

4.1 Test Environment Configurations

In order for a successful test there needs to be a controlled environment that resembles as close as possible the Production environment for the application. Configuration Management (CM) must exist on the Test environment so each version of all associated applications is known during every test cycle. Any changes during a test cycle can impact the outcome of the individual test cases, and the overall test cycle and invalidate the test phase. CM will be responsible for software builds, setup, and configuration.

4.2 Base System Hardware

System Hardware Resources		
Resource	**Quantity**	**Name and Type**
Database Server	1	vista.raygroupintl.com, Cache on Windows 2008 Server
Database Name	1	VistA
Test Repository	1	kthlnkeating/tests, git repository
Server Name	1	Github.com
Test Development PCs	2	Windows XP machines with local Cache installation, also with Virtual Box Ubuntu GT-M installations

5 Staffing and Training Needs

Please reference the Project Communications and Resource Plan for all current staffing plans. Additional staffing and training will be addressed as refactoring and testing progresses.

6 Test Metrics

Metrics are a system of parameters or methods for quantitative and periodic assessment of a process that is to be measured. Test metrics may include, but are not limited to:

- At least one acceptance test per refactored module
- Test time to run
- Time to re-test
- Test run frequency
- Defect Identification
- 100% of the test cases must pass
- Percentage of test cases executed
- Percentage of test cases resulting in defect detection
- Number of defects attributed to test case/test script creation
- Number of defects per requirement
- Percentage of defects identified; listed by cause and severity

7 Risks and Constraints

The Project Manager defines the level of risk deemed tolerable for the project, how risk is managed, who is responsible for risk activities, the amounts of time that are allotted to risk activities, and how risk findings are communicated.

The Test Team reports risks for evaluation. Risks are tracked as impediments. Implementing early testing during iterative design and development is a risk mitigation strategy for the overall project.

Appendix A: Test Type Definitions

Testers use “test types” to validate the system or application under test. Simply put, test types are test techniques used to exercise the system or application. This section presents a minimum set of test types that can be performed, listed by the development phase when they would occur, and provides definitions for each of these test types. The Tester, in consultation with the Chief Architect, selects the test types best suited to the system or application being tested. A minimum set of test types is suggested here; more tests may be added at the discretion of the Technical Team.

A-1: Test Types by Development Phase

Table A-1 presents a listing of test types that may be utilized during the product build, independent testing, operational readiness testing (ORT), and initial operating capability (IOC) testing.

Table A-1: Test Types by Development Phase

Type of Test	Product Build Testing	Independent Testing	OSEHRA
Access Control Testing	X		
Build Verification Testing		X	
Installation Testing	X		
Integration Testing	X	X	
Security Testing	X		
Smoke Testing	X	X	
Usability Testing			X
User Functionality Testing	X		
User Interface Testing	X		

A-2 Test Type Definitions

The test types listed in Table A-1 are defined in Table A-2.

Table A-2: Test Type Definitions

Test Type	Definition
Access Control Testing	A type of testing that attests that the target-of-test data (or systems) are accessible only to those actors for which they are intended, as defined by user stories. Access control testing verifies that access to the system is controlled and that unwanted or unauthorized access is prohibited. This test is implemented and executed on various targets-of-test.
Build Verification Testing (Prerequisite: Smoke Test)	A type of testing performed for each new build, comparing the baseline with the actual object properties in the current build. The output from this test indicates what object properties have changed or don't meet the requirements. Together with the smoke test, the build verification test may be utilized by projects to determine if additional functional testing is appropriate for a given build or if a build is ready for production.
Installation Testing	A type of testing that verifies that the application or system installs as intended on different hardware and software configurations, and under different conditions (e.g., a new installation, an upgrade, and a complete or custom installation). Installation testing may also measure the ease with which an application or system can be successfully installed, typically measured in terms of the average amount of person-hours required for a trained operator or hardware engineer to perform the installation. Part of this installation test is to perform an uninstall. As a result of this uninstall, the system, application and database should return to the state prior to the install.
Security Testing	A type of test that validates security requirements and ensures readiness for the independent testing performed by the Security Assessment team as required by the Test and Certification process.
Smoke Test	A type of testing that ensures that an application or system is stable enough to enter testing. It is usually a subset of the overall set of tests, preferably automated, that touches parts of the system in at least a cursory way.

Appendix B: Problem List Testing

The Problem List application can be tested through the execution of the prescribed test cases from the NIST. In addition to the NIST the CPRS Problem List User Manual details GUI tests which can be performed to validate the refactored Problem List.

- http://healthcare.nist.gov/use_testing/effective_requirements.html Criteria 170.302(c) Maintain up-to-date Problem List. http://healthcare.nist.gov/docs/170.302.c_problemlist_v1.1.pdf
- ***CPRS Problem List User*** Manual ,December 2004, Health Data Systems Veteran Health Administration, www.va.gov/vdl/documents/Clinical/CPRS-Problem_List/gmplum.doc

In refactoring it is important to note that the functionality of the Problem List implementation is not changing. Rather, the code base is being improved upon structurally, but the requirements established for the original Problem List module are not changing. The requirements mentioned previously in this document are only those established internally by the project's Technical Team for designing and executing a successful refactoring strategy.

Both GUI testing through CPRS and testing through the Intersystems Cache Terminal will be performed.