Distributed Incident Management System (DIMS)

Software System Test Plan (SSTP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| APPROVED BY: |  | |  |  |
|  | Dave Dittrich, Principle Investigator |  |  | Date |
| APPROVED BY: |  | |  |  |
|  | Mickey Ross, Program Manager |  |  | Date |
| APPROVED BY: |  | |  |  |
|  | Audrey Jeffery, SQE Consultant |  |  | Date |

Change History

|  |  |
| --- | --- |
| Version | Description |
| 1.5 | Initial Release |

Table of Contents

1. Introduction 4

2. Definitions, Acronyms and Abbreviations 4

3. References 5

4. Software Test Strategy 5

4.1 Overview 5

4.2 Scope of Testing 5

4.3 Assumptions 8

4.4 Risks to Test Schedule 9

4.4.1 Identified Schedule Risk(s) 9

4.4.2 Mitigation for Schedule Risk(s) 10

4.5 Turnover to SQE 10

4.6 Test Readiness Reviews 10

4.7 Test Environments Error! Bookmark not defined.

4.8 Software System Test Strategy (General) 10

4.8.1 Test Types 10

4.8.2 Test Strategy 12

4.8.3 End User/Beta Test Strategy 13

4.8.4 Test Cases 13

4.8.5 Reporting Test Status 13

4.9 Traceability 13

4.10 Tools Analysis 13

4.11 Acceptance Criteria 13

4.12 Time Estimate 13

5. Deliverables 13

# Introduction

This Software System Test Plan documents the objectives, approach, and scope/limitations of the testing and evaluation of the Distributed Incident Management System (DIMS). It describes the logistics for executing the plan, including test environment needs, high level schedule, and resource mapping. It also identifies technical and schedule risks and mitigation plans.

This plan will be utilized by the project team to properly scope, manage, and comprehend test efforts.

This test plan is specific to software, as defined within the statement of work.

Project Description:

The primary mission objectives for the DIMS system are operational in nature, focused on facilitating the exchange of operational intelligence and applying this intelligence to more efficiently respond and recover from cyber compromise. The secondary mission objectives are to create a framework in which tools to support the primary mission objectives can more quickly and easily be integrated and brought to bear against advancing techniques on the attacker side of the equation.

The DIMS project is intended to take this semi-automated sharing of structured threat information, building on the success of the Public Regional Information Security Event Monitoring (PRISEM) project and leveraging an existing community of operational security professionals known as Ops-Trust, and scale it to the next level. The intent of this software project is to allow for near real-time sharing of critical alerts and structured threat information that will allow each contributing party to receive information, alerts and data, analyze the data, and respond appropriately and in a timely manner through one user-friendly web application, DIMS.

Working with the use cases defined by MITRE and PRISEM users, building the features necessary to simplify structured information sharing, and operationalizing these within these existing communities, will allow DIMS to fill existing gaps in capabilities and support existing missions that are slowed down today by many complicated, manual processes.

The changes to existing systems consists of seamless integration of the three current systems into a single web application that enables each system to contribute to the data warehouse of information concerning threats, alerts, attacks and suspect or compromised user terminals within the infrastructure. Additionally, the integrated systems will be able to share and retrieve data, visually observe alerts through color coded visual indicators, while retaining the existing functionality of their current system

# Definitions, Acronyms and Abbreviations

Give the definiti***o***ns, acronyms, and abbreviations.

1. Definitions, Acronyms and Abbreviations

| Term | Definition |
| --- | --- |
| Feature | A prominent characteristic that describes a new design. |
| System Requirements | DIMS System Requirements and Concept of Operations |
| DIMS | Distributed Incident Management System |
| PRISEM | The Public Regional Information Security Event Management |
| OpsTrust | Operations Security Trust (see <http://ops-trust.net>) |
| IOCs | Indicators of Compromise |
| AS | Autonomous System |
| CIF | Collective Intelligence Framework |
| CSIRTs | Computer Security Incident Response Teams |
| SLTT | State, Local, Territorial, and Tribal government entities |
| US-CERT | US-CERT |
| STIX | Structured Threat Information eXpression |
| SIEM | Security Information Event Management |
| URL | Universal Resource Locator |
| SITREP | Situational Awareness Report |
| CIDR | Classless Internet Domain Routing |
| AMQP | Advanced Message Queuing Protocol |
| CoA | Course of Action |
| .csv, CSV | Comma Separated Value file |
| IP | Internet Protocol |
| KPP | Key Performance Parameters |
| UI | User Interface |
| TAXII | Trusted Automated Exchange of Indicator Information |

# References

1. References

| Document | Description |
| --- | --- |
| DIMS\_Architectural\_Design\_v0.1\_20140330\_DD | Overall architecture of each existing system and the integrated solution |
| DIMS\_Systems\_Requirements\_and\_CONOPS\_v0.1\_201403\_DD | Use Cases and requirement narrative |
| SOW APL-UW-dd-20130905 | Explicit statements of expected work |

# Software Test Strategy

## Overview

Delivery of software for testing will be conducted in the following manner:

* Software for the project will be developed using the Agile Methodology and will therefore be delivered incrementally, based on completed functionality for each Sprint.
* Each Sprint cycle is anticipated to be two to four (2-4) weeks.
* A final feature complete build will be delivered at the completion of development.
* Testing efforts will be structured to test the delivered functionality of each Sprint’s delivered software module(s).
* Feature complete builds will be tested as integrated modules.
* Iterations of testing will be conducted as builds are delivered to address bug fixes as well as delivered features or functionality.

## Scope of Testing

The following are features/functions that will be tested, but does not represent the order in which it will be tested, as testing is based on each Sprint’s delivered functionality. The scope of this project’s testing includes the following:

* Integration of existing systems (PRISEM and OpsTrust) with DIMS
* Human-centered User Interface
* User Roles
* Role based access controls
* User attribute storage
  + Contact information
  + Preferences
  + Site-specific information
    - CIDR blocks
    - Top level DNS Domains

User Tasks

* Search CIF
* Search historic network flow records
* Search historic event records
* Import/export Observables and/or Indicators of Compromise (IOCs)

User notifications

* Email threads of interest
* Messages
* Tailored Reports
* Activity of Interest Alerts

User Experience

* User Authentication
* Single Sign-on
* Browser Based Dashboard:
  + Google Chrome
  + Ubuntu
  + (Other browers/operating systems as resources permit)
* Keep track of multiple:
  + Incidents
  + Campaigns
  + Sector-specific threat activities
  + Other ad-hoc groupings of security information
* Detection of IP address(es) associated with CIDR block(s)
* Detection of Domain names associated with top level Domain(s)
* Trigger workflow processes, e.g.
  + Send an alert to a user
  + Generate a schedule report
  + Trigger a search of available data
* Visual Alerts
* Set Thresholds
* Real-time communication and alerts
* Analysis tools within application
* Structured/Unstructured data capable database
* Database connectivity
* Database queries
  + Iterative
  + Recursive
* Linux command line
* Web services
  + Verify request input parameters
  + Verify expected input parameters
  + Verify unexpected input parameters (i.e. alpha inputs when only numeric inputs are expected-negative testing)
  + Verify expected response returned
  + Verify response format and content
  + Verify exceptions are handled properly
* Data mining
* Data visualization
* IM/Chat interface
* Import of machine-parseable bundles of Structured Threat Information eXpression (STIX) data from US-CERT and trusted portals, i.e. Ops-Trust, or Beadwindow
* Export of machine-parseable bundles of Structured Threat Information eXpression (STIX) data to US-CERT and trusted portals, i.e. Ops-Trust, or Beadwindow
* Verification that data transferred between systems is encrypted
* Verification that transferred data is signed and time stamped
* Verification of proper storage of imported structured and unstructured data
  + Properly inserted into database
  + Integrity of imported data
  + Intended format of inserted data
  + Retrievability of imported data
* Key Performance Parameters (KPPs) for DIMS components:
  + Backend Databases
    - Verify timeliness of data availability
    - Verify timeliness of queries
  + Web Application Service
    - Verify request and response times
  + Data Integration & User Tools
    - Verify seamless integration of input data and its availability in User Tools
  + Vertical and lateral information sharing
* Reports generation
  + Verify timeliness of report generation
  + Verify content of reports based on specific report filters
  + Verify format of generated reports
  + Summarization of any/all aggregate data presented to user:
    - Context data
    - Start/End date and time
    - Total within “Friend” population (Systems)
      * Breakdown of Participants
    - Total outside of “Friend” population
      * Breakdown by country/AS/IP address
    - Total “not-friend” population (Systems) (known foes)
      * Breakdown by country/AS/IP address
    - Aggregate summarization of *X* number of IPs
      * Drill-down capabilities
* Transmission of structured data files
  + Verify transmission of various file types
* Receipt of structured data files from a variety of sources:
  + Email attachments
  + Encrypted email attachments
  + CIF feed
  + AMQP message bus
  + Upload from user’s workstation via DIMS dashboard client
  + From Tupelo client
  + Other asynchronous automated mechanism (what might those be?)
  + From other command line mechanism (what might those be?)
* Automated processing of the structured input/output data files:
  + IOCs
  + Observables
  + Course of Action
  + Situational Awareness Reports
  + Incident Reports
* Manual receipt (entry) of IOCs (STIX) from member(s) of trust community
* Processing of IOCs regardless of receipt method/means (manual/semi-automated) or origin (internal/external)
* Automated IOC Sharing
  + Verify internal to external IOC sharing
  + Verify external to internal IOC sharing
  + Verify timeliness of internal to external IOC sharing
  + Verify timeliness of external to internal IOC sharing
* Real-time Situational Awareness reporting
  + Verify reporting is done in real-time
* Alternate response actions
  + Verify generation of alternate response actions
  + Verify timeliness of alternate response actions
* Host level triage
  + Verify the ability to link network level Indicators and observables-
    - IP address (es)
    - Domains
    - URLS

-To host level observables

* + - Registry Keys and values
    - File names
    - Cryptographic hashes of files
  + Verify ability to search on host level observables
* Verification that network flow history, event history and attacker context history flows into the triage process within the integrated system
* Cross organizational correlation of events
* Forensic information linking (linking host computer data with network traffic data)
* The anonymization of security event data
* Redacted and un-redacted data sharing

The delivery of functionality will be based on development cycles and delivery schedule; the above scope of testing does not indicate what functionality will be delivered in which Sprint.

NOTE: Testing or Regression testing of existing systems (i.e., PRISEM or Ops-Trust) is not included within the scope of testing.

## Assumptions

Testing makes the following assumptions:

1. Existing components, i.e. PRISEM and OpsTrust, work as expected
2. Users of existing functionality will verify related tasks within DIMS application
3. Functionality delivered in each Sprint is modular and is not dependent upon any other module(s)
4. A working prototype will be available for testing
5. Input data will be provided for testing
6. Specific format of expected output/data is supplied

## Risks to Test Schedule

### Identified Schedule Risk(s)

Risks negatively affect the testing timeline and are listed below.

1. Test Schedule Risk – Mitigation Table

| Risk | Description | Impact | Mitigation |
| --- | --- | --- | --- |
| 1 | Functionality for a Sprint is delivered late | Delay in start of testing for that Sprint  Delay in test complete date | Schedule of delivery of functionality for each Sprint is developed and adhered to |
| 2 | Undocumented changes | Delay due to investigation of changes (working as designed or defect?) | All changes are documented, i.e., in build manifest or build notes. |
| 3 | Feature Creep (Increase in scope) | Increase in time to analyze additional scope, create test cases and execute will delay test completion | Locked requirements |
| 4 | Unresolved roadblocks (blocking bugs, defects, trackers) | Delay in test execution  Delay in test completion | Bug fix prioritization  Showstoppers or critical defects fixes delivered in a timely manner |
| 5 | Lack of resource availability | Additional work load and time to create and execute testing tasks with fewer resources  Delay in test completion | 1. Reallocation of resources to adequately execute testing tasks and deliverables  2. Decrease scope of testing to accommodate available resources. |
| 6 | Changes in project schedule | Decrease in project timeline will cause shorter testing timeline  Higher incidence of missed defects | Realistic project milestones and delivery dates |
| 7 | Third Party dependencies | Delay in receiving third party deliverables will block progress and cause delays | Ensure all groups or teams deliver their inputs or deliverables in a timely manner |
| 8 | Software Functionality dependencies (functionality delivered in a Sprint is dependent upon other, undelivered functionality) | Inability to test delivered functionality  Delay in test completion for the Sprint | Delivery of testable code/functionality for each Sprint that is executable/testable autonomously |
| 9 | Ambiguous or incomplete requirements | Incorrect interpretation of requirements may cause a delay due to re-writing test cases, wrong approach to testing, missed requirement | Clear, concise, testable requirements |

### Mitigation for Schedule Risk(s)

Please see Table 3 above for mitigation of schedule risks.

## Turnover to SQE

Turn over to SQE will include:

* Manifest of delivered functionality
* Release notes
* Delivery of software will be downloadable/accessible from Secure Wiki
* Identified build/Sprint number per development schedule
* Bi-Weekly builds

## Test Readiness Reviews

The purpose of the Test Readiness Review is to ensure the readiness of the following:

1. Test cases for execution and for test coverage of requirements
2. Software to be tested
3. Test environment
4. Any testing tools utilized for testing
5. Issue tracking

Test Readiness review will be conducted by the following core team members:

* Principle Investigator
* Program Manager
* Software Quality Engineer

## Software System Test Strategy (General)

The software system test strategy we will employ is compromised of two high-level sets of tests.

The first set of test will be developed as the system is being designed using the Agile coding methodology. Development and pre-release testing will be performed during “sprints” (cycles of development anticipated to be on 2 week time frames). These tests are centered on development and take place within the project team. The project team will extract user stories from the use cases and in turn create epics derived from the user stories. Tests will be written for the user stories and determine the necessary development. This is essentially a test-driven approach that pushes development from one sprint to the next.

The other set of tests will be performed with interaction from the stakeholders and will entail an environment in which the stakeholders will, at determinate intervals, access the system and execute one or more user stories. This environment, set up specifically for stakeholders, will enable them to interact with the system at their discretion and direct their feedback to the project team about the functionality delivered within their environment.

### Test Types

The following table identifies which types of testing will be utilized for the project. Within each test type is an explanation of how that type will be used and a justification of how it meets the scope of testing for the project.

1. Test Types

|  |  |  |
| --- | --- | --- |
| **Test Type** | **Test Explanation** | **Justification** |
| Functional Testing | Verification of functionality of DIMS | Compliance with user centered design and function |
| User Interface Testing | The design is based upon an explicit understanding of users, tasks and environments.  The design addresses the whole user experience. | In addition to the look and feel of the web application, the user interface must consider each user type and their tasks, for ease of use. |
| Performance Testing | DB queries  Concurrent users (logged in system)  Concurrent user queries  Concurrent user report generation | -Determines a baseline of the system performance  -Identifies any bottlenecks in retrieving or displaying data  -Helps to determine the load the application can handle at a given threshold |
| Security Testing | User access  Single Sign-on | Verify at a minimum that security is adequate, user passwords are encrypted and the user’s single sign on to DIMS gives access to each component within DIMS |
| Deployment Testing | Replicates the process and components and push of a software deployment and verifies the process, all necessary components have been included and that there are no roadblocks to push (deploy) the software to external systems’ servers | There is a server that will provision, configure and administer DIMS components and then push the DIMS components to each user’s server automatically |

### Test Strategy

User stories will be converted into test cases that exercise the various portions of the user interface and functionality of the software in light of each representative user’s tasks. This will be accomplished through a breakdown of the user story as exampled below.

The strategy for testing each user story is to:

1. Translate each user story into an end to end test case
2. Determine if additional use cases are needed to test variations of a use case
3. Parse the test case(s) into testable sections based on:
   1. The portion of software utilized in each section of the user story
   2. The pre-determined functionality delivered in each sprint
4. Enter that test case (each parsed portion) based on the format described above into the Test Case Management Tool indicated in the Tools section, 4.10
5. Verify that each parsed portion of a user story has its own test case
6. At the end of each sprint, the relevant/applicable test case(s) will be executed to verify its functionality, based on:
   1. The expected functionality delivered for that sprint
   2. The expected behavior of the software.
7. Any deviations from the expected behavior will be:
   1. Entered into defect tracking tool (as noted in the tools section, 4.10)
   2. Reviewed/evaluated by the core team
   3. Dispositioned
      1. Fix issue
      2. Reject-Working as Designed (issue will be closed with justification)
      3. Need more information
8. Once an issue has been fixed:
   1. Developer assigned to fix the issue:
      1. Enters a fix note indicating
         1. Root cause of issue
         2. What was discovered during investigation/fixing the issue
         3. What was fixed
         4. How the issue was fixed
      2. Sets the defect to Ready for Test
      3. Assigns the issue back to submitting tester for verification
   2. Tester assigned to verify issue:
      1. Verifies the issue
      2. Enters Pass/Fail note indicating
         1. Steps utilized to verify the issue
         2. Any input data utilized to verify the issue
         3. Outcome of verification (Pass/Fail)

### End User Acceptance Test Strategy

Software produced within each sprint will be accessible to stakeholders to remotely access and interact with the functionality available in the interim builds of the DIMS software. Feedback from stakeholders about the delivered software will be evaluated and processed into functional changes to the software to improve the user experience and task execution.

End user acceptance testing will be conducted throughout the development process, and will utilize a replica or prototype of the DIMS System to exercise the software in an environment that will closely resemble the finished product. Users will exercise the system much in the same way that they would in order to accomplish the various tasks in a given day. This environment will receive incremental builds of software where by the end user can interact with the software as it is being built, and give vital feedback to help guide and direct the development efforts. This essential feedback will communicate to the development when the various functionalities have reached maturity.

### Test Cases

Testing will be based on requirements, user stories and use cases. Delivered features during each sprint will be tested as they are made available. Individual test cases will be developed in support of the Sprint. (See Figure 1 for an example of a test case.)

The test cases will be reviewed and approved in accordance with the core team’s current test case review policy, prior to conducting testing. Evidence of the review and approval, along with copies of the test cases approved is documented. The test case review ensures: the objectives in the DIMS Software System Test Plan are met, the accuracy of requirement references, the completeness and consistency of test cases, and that the required data elements are contained in each test case.

Test cases will contain the following minimum data elements: Test Case Name/Title, Author, Cross reference to the applicable software release, Test Objective, Cross-reference to the requirement if applicable, test set up, test steps and expected outcome.

* Test Case Name/Title
* Author
* Cross reference to the applicable software release, i.e. build number/version
* Test Objective
* Cross-reference to the requirement if applicable
* Test Setup - set up information necessary to conduct the test case, including detailed hardware configuration and specific hardware components
* Test Steps - describing the action(s) or sequence of actions to perform the
* Expected Result - indicating the desired behavior by the system when subject to the action in the Test Step

The following fields are included and left blank for the initial test cases. The fields are then populated during test case execution:

* Actual test results - the real time recording of observed results during a test case execution. Note: Any raw data collected during the test execution will be attached to the Test Execution Report
* Status of the test - Pass or Fail
* Test Unit Configuration – the software versions and hardware configurations for all modules incorporated into the test as applicable.

### Reporting Test Status

Test status will be delivered at the end of each Sprint’s testing effort. To include the following:

* Total number of test cases for the sprint
* Total number of test cases executed
* Total number of Pass/Fail, respectively
* Total number of defects for the Sprint
* Percent complete

## Test Environments

The DIMS team will use two test environments (i.e., two sets of DIMS service components), one for development and the other for end-user testing.

For development purposes using the agile model, an instance of the DIMS system components will be used as part of the Continuous Integration build/deploy/test cycles. This environment will consist of dedicated severs for DNS, web portal, single-sign on, CIF, AMQP Broker, data processing services, DIMS data back end, etc. The environment will also include Open VPN remote access, a browser-based dashboard UI and Linux command line front end. This environment will be refreshed every two weeks, according to the development sprint cycles, with interim builds of software containing new features.

For end-user testing a separate instantiation of the DIMS system components will be made available on-site at the UW and/or deployed on-site in the stakeholder’s computing facilities as desired by the specific stakeholder. This environment will include all of the components listed above, but dedicated to end-user testing and feedback. This environment will be refreshed with build iterations at a specified rate not necessarily consistent with the development build cycles. Deployment of new feature builds will depend on the timeframe in which end users are able to use the system and return feedback to the project team to help drive their development efforts.

The test environments will also consist of:

1. Input data from the following stakeholders:

* Ops-Trust
* Beadwindow
* US-CERT
* Other SLTT groups

1. A prototype of the DIMS system including:
   1. Database(s)
   2. User interface
   3. Supported browser(s)
   4. A platform-independent "API" or "middleware" model

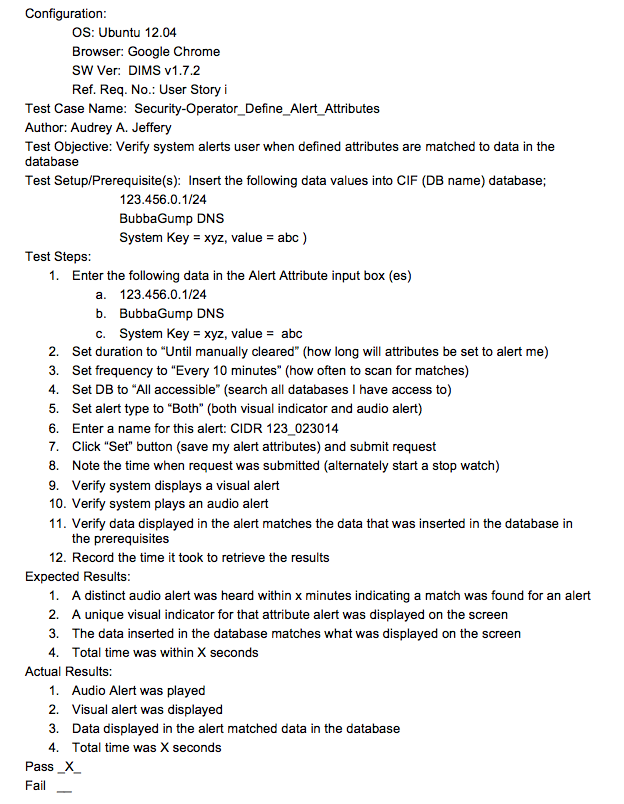


Figure - Example Test Case

## Traceability

Test cases executed will be traced to requirements and/or user stories as stated in the System Requirements document.

## Tools Analysis

Testing tools will be utilized to effectively document, track and reuse tests, and to provide traceability. The exact tools that will be used during testing have not been selected; however, the tools currently under evaluation and consideration have been provided in the table below. The table below represents the tools that made it past the first stage of evaluation, which included a list of 21 tools that were evaluated for their features, cost, simplicity of Use, and appropriateness for the DIMS software development and testing needs. The six tools below were assessed and recommended for DIMS consideration based on the previously discussed criteria. The light green font color highlights the benefits of the tool and the red font color indicates a negative consideration of the tool.

Testing tools and their intended use will be documented accordingly.

All testing tools utilized during testing will be documented within the Test Report.

Test Management tool:

|  |  |  |  |
| --- | --- | --- | --- |
| [Behave for JIRA](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-BehaveforJIRA" \t "_blank) | Behave for JIRA is a powerful and easy to use agile testing and requirements discovery tool. It allows users to take TDD to the next step, Behavior Driven Development. Specific needs of a software product are expressed as acceptance criteria on User Stories, invaluable for discovering the real requirements through discussion. The acceptance criteria can be automated as tests using Cucumber to provide instant feedback on software development progress. Can be downloaded and installed inside the Atlassian application. If you use Scenario Outlines in your test, the report will show in an "Unknown" state. This is not a problem with this plugin, but instead with how Cucumber is doing the JSON reports. If you use the HTML reports currently provided by Cucumber, you will love this plugin. | Feature Highlights:  Add scenarios to individual User stories (or any JIRA Issue) as acceptance tests; scenarios are written in a natural language called Gherkin (Given When Then) so any one can write them; simple test management with the choice between manual and automated testing; scenarios/acceptance tests can be automated using Cucumber, a popular open source testing tool. No vendor lock in; export tests from JIRA with Cucumber; selectively run automated tests based on if the parent JIRA issue is closed or in progress. No $10 starter license. Technical support for customers using the JIRA Plugin and Maven integrations is $100 for 10 users. | Cost, simplicity, and integration, are the most notable.  Recommended for DIMS. |

|  |  |  |  |
| --- | --- | --- | --- |
| [Zephyr for JIRA](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-ZephyrforJIRA" \t "_blank) | Zephyr provides end-to-end management of the testing lifecycle including resources, releases and sprints, test cases, scheduling, test execution, defects, documents, collaboration and all aspects of reporting and metrics in real-time. Built on an extensible management platform, Zephyr allows users to leverage their existing tool investments by providing seamless integration with popular defect tracking systems like JIRA and interoperating with commercial, open source or home grown automation tools. As a native add-on built exclusively for the JIRA 5 and 6 platforms, Zephyr for JIRA completes end-to-end project management in JIRA by adding testing to the overall planning, development, bug tracking and reporting process. Tests are written for a JIRA project. They can be further organized and grouped by Versions, Components and Labels that have been set up for that project. A test can belong to one or more of these. These tests can also be searched using the "Issue Navigator" and bulk changes can be made on them.  Just like with any other issue, you can do the following with tests: Create; Import; Clone; Search; Bulk Modify; Link; Label; Share; Export; Print; Vote; and Comment on them. Zephyr for JIRA is best suited for project teams that want to integrate testing into their workflow. [ZAPI 1.0](https://marketplace.atlassian.com/plugins/com.thed.zephyr.zapi" \t "_blank) is a powerful new add-on to Zephyr for JIRA, allowing access to its testing data programmatically via RESTful APIs. Zephyr for JIRA is also very useful for nimble teams that just need to create and execute tests in an ad hoc manner and keep track of status at a high level without too much detailed planning.   Metrics can be broken down by projects and versions, and grouped by user, test cycles and components. They can be displayed alongside each other in a dashboard to provide a comprehensive testing view for a particular version or project. These self-updating charts always provide a near real-time view into the quality of your software releases giving you the confidence to make important business decisions based on them. | Detailed descriptions in tests - like individual test steps, test data and expected results are easily entered in each test with nifty inline edit capabilities, drag-n-drop reordering of steps and deletion. File and screenshots can be attached to every test. Historical information about tests is stored.  Tests are easily linked to requirements, be they Enhancements, Tasks, Epics, Stories or other issue types. This allows for tracking test coverage and traceability. Look at any particular test and you instantly know which requirements it is linked to, what its execution status is and which defects were filed against it.  Create tests with Bonfire  Zephyr for JIRA integrates with Atlassian's Bonfire, allowing users of Bonfire to create tests directly from the web application they are testing. Tests are created with detailed descriptions, instant screenshots and dynamic variables. Tests can also be created and executed as part of Bonfire Test Sessions and existing tests can be added to a new Test Session.  Test metrics are captured and displayed in Open Social based gadgets that are accessed from JIRA's Gadget Directory. Test Distribution and Test Execution gadgets can be added to any dashboard and customized across various parameters allowing for specificity in the data being displayed.    Zephyr for JIRA is available only for the Download version of JIRA 5.x/6.x and is very affordable. The license model has been structured to align with JIRA's license tiers - which means you have to purchase the same # of licenses as what you have on your JIRA 5.x/6.x server. All licenses are perpetual and include 12 months of software maintenance and support. License tiers cannot be combined nor can we do custom tiers. 10 Users, $10, Perpetual, includes 1st year of software maintenance; 25 Users, $750, Perpetual, includes 1st year of software maintenance. As users double, the price doubles, but the software maintenance is the same. | Cost, simplicity, and integration, are the most notable.  Recommended for DIMS |

|  |  |  |  |
| --- | --- | --- | --- |
| [QMetry](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-QMetry" \t "_blank) | JIRA Plug-in for QMetry: QMetry is a comprehensive Test Management Platform equipped with JIRA integration. The Metric-Rich Dashboard enables real time views into the quality of your software application. QMetry integrates with JIRA to allow testers to enter bugs directly into JIRA from QMetry. These bugs will pull the relevant test case information into the defect to automatically provide steps to reproduce the bug, and testers can provide additional notes and attachments (log files, screenshots, etc.). QMetry can also link existing issues to Test Cases or Requirements or be setup to automatically pull JIRA issues to create new Test Cases or Requirements based on advanced search criteria.  In addition, with the QMetry plug-in JIRA users can have seamless access to QMetry test cases and requirements right from their JIRA user interface.  More details on QMetry integration [here](https://plugins.atlassian.com/plugin/details/14077" \t "_blank). | The plug-in helps to: Capture all relevant details from the JIRA issue to the test case, or requirement in QMetry; browse the test cases from JIRA and readily create a defect with one click. Steps to reproduce the defects migrate from test case to defect automatically; associate a JIRA defect to QMetry test case or requirement right from JIRA; search and find test cases or requirements from within JIRA; browse QMetry projects, releases and builds; the name: JIRA Plug-in for QMetry Version 2.2, QMetry Product Versions: 4.0 and above is free for licensed QMetry Enterprise users. | The JIRA plug-in is FREE to all QMetry Professional $50/user/month billed annually and Enterprise customers $75/users/month      Recommended for possible consideration for DIMS. |

|  |  |  |  |
| --- | --- | --- | --- |
| [SpiraTest](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-SpiraTest" \t "_blank) | Complete Quality Assurance solution that manages requirements, tests, bugs and issues in one environment, with complete traceability from inception to completion. Free three user 30-day trial hosted account; 1 concurrent user, software and hosting $5.99/month; 3 concurrent users, software and hosting; $69.99/month; 5 concurrent users, software and hosting $99.99/month; 10 concurrent users, software and hosting $179.99/month  More details on SpiraTest integration [here](http://www.inflectra.com/SpiraTest/Default.aspx" \t "_blank). | With the [JIRA connector](http://www.inflectra.com/SpiraTest/Downloads.aspx" \t "_blank) activated, any defects logged during a test execution get routed to JIRA using the SOAP API. Any changes to these issues in JIRA then get synchronized back into SpiraTest using our SOAP API. For the tester, SpiraTest provides the ability to execute containing predefined groups of test cases (along with their test steps) so that the testers can follow the instructions and determine if the system being tested behaves as expected. Any deviations from expected behavior can then be recorded as defects and managed in the built-in [defect tracking](http://www.inflectra.com/SpiraTest/Highlights.aspx?feature=Defect-Tracking" \t "_blank) module. This allows quality assurance software testing activities to be performed exactly as specified by the project manager. | SpiraTest Hosted Service offers the possibility to synchronize with a local JIRA via a desktop tool. With the help of Inflectra support we got it running.  Have a complete test management solution for the time needed - Working synchronization with JIRA - Easy-to-install desktop utility for synchronization - low setup and operating costs.      Recommended for possible consideration for DIMS |
| [Test Collab](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-TestCollab" \t "_blank) | Test Collab is web based test case management tool with simplest and easy to understand AJAX enabled UI. It offers great features like issue manager integrations, time forecasting, time tracking and in-built reporting. $25/1-4 users/month or $250/year/user; $23/5-9 users/month or $230/year/user; $20/10-19 users/month or $200/year/user | It supports issue manager integration with famous bug tracking tools like JIRA, Redmine, Unfuddle, Mantis, Lighthouse, and FogBugz. Test Collab is available in both: SaaS and self-hosted editions. A default Jira settings form has 8 pre-entered fields. You can click the test button at bottom to make sure that Test Collab is able to create a sample bug in Jira. | Confident Test Collab can manage test cases with simplicity however, the cost is not the least among others.    Recommended for possible consideration for DIMS |

|  |  |  |  |
| --- | --- | --- | --- |
| [Testuff](https://confluence.atlassian.com/display/JIRAKB/Integrate+JIRA+With+A+Test+Case+Manager" \l "IntegrateJIRAWithATestCaseManager-Testuff" \t "_blank) | Testuff is an **on-demand management service** that [integrates with JIRA](http://www.testuff.com/help/jira" \t "_blank). Amongst the standard features, it also includes a test video recorder. **The recorder captures bugs as they happen during testing and allows you to send the videos to your developers**, making the "works for me" excuse obsolete. All in all Testuff is very easy and intuitive to use, even a beginner tester could get fully working on it within an hour. **All Packages Include: 24×7 Support; Maintenance; Hourly Backups; Unlimited Projects; Tests; Free Automatic Upgrades; Unlimited Storage; Secure Access over SSL;$27/user/month; $270/user/year** | Exports bugs found during test runs directly to the relevant project in JIRA with your JIRA user. The integration is done by the desktop client. Therefore, it should also work if you host JIRA on an internal server. Our server doesn’t need to access your bug tracker server and you don’t need to change anything in your security or firewall configuration to make it work. | Unlimited testers, tests, projects, defect reports; Highly secure, authenticated and fully backed-up environment; No need for a server, get started right away;  Highly intuitive, easy to use; Simple license management although the cost adds up quickly with the $270/user/year cost for this on-demand test management service.        Recommended for possible consideration for DIMS |

## Acceptance Criteria

Acceptance criteria for release of this product:

* All functionality defined in the System Requirements document are verified and all associated test cases have passed
* All requirements have associated test cases
* Issues identified during testing are verified as fixed or deferred, with justification
* Integration testing is verified
* The software does not crash, hang, or have abnormal behavior during test execution
* There are no undefined/undocumented features in the released system

## Time Estimate

Estimation of total testing time is yet to be determined.

Testing is conducted in conjunction with software development and is dependent upon the complexity of the functionality delivered in each sprint.

Test completion will be as per project schedule.

# Deliverables

1. List of SW Test Deliverables

| Deliverable Name | Delivery Date |
| --- | --- |
| DIMS Test Plan Draft | May 9, 2014 |
| DIMS Test Report Draft | September 11, 2014 |
| DIMS Test Plan | February 8, 2015 |
| DIMS Test Report | June 5, 2015 |