**SySTemantics Test Approach Document**

By

Dixita Sharegar

Bhargav Uppalapati

Siva Chintapalli

**Sponsor**

**Anthony Giorgio**

**Poughkeepsie, NY**

**April 2016**

Contents

[**1.0** **INTRODUCTION** 3](#_Toc447820697)

[**2.0** **SCOPE** 3](#_Toc447820698)

[**3.0** **Testing Strategy** 3](#_Toc447820699)

[**3.1** **Unit Testing** 3](#_Toc447820700)

[**3.2** **System and Integration Testing** 4](#_Toc447820701)

[**3.3** **Performance Testing** 4](#_Toc447820702)

[**3.4** **Function Testing** 4](#_Toc447820703)

[**3.5** **Configuration Testing** 5](#_Toc447820704)

[**3.6** **Regression Testing** 5](#_Toc447820705)

[**4.0** **Hardware Requirements** 5](#_Toc447820706)

[**5.0** **Environment Requirements** 5](#_Toc447820707)

[**6.0** **Test Schedule** 5](#_Toc447820708)

[**7.0** **Control Procedures** 6](#_Toc447820709)

[**7.1** **Problem Reporting** 6](#_Toc447820710)

[**8.0** **Features to Be Tested** 6](#_Toc447820711)

[**9.0** **Schedules** 6](#_Toc447820712)

[**10.0** **Risks/Assumptions** 6](#_Toc447820713)

[**11.0** **Approvals** 6](#_Toc447820714)

### **INTRODUCTION**

A Metrics collector is a console based application which runs on the Linux-based operating system. The application incorporates *CPU* stats, memory stats, Network stats. It helps users of the system to manage the CPU throughput, and view network and memory stats allocated to the applications.

Keeping track of all the system metrics without a proper application is hard. It is also very difficult to remember the commands to view all the system statistics. The statistics are logged into a database frequently which are analyzed and displayed to the user to keep track of system performance.

### **SCOPE**

This Test Plan describes the integration and system tests that will be conducted on the SySTemantics following integration of the components identified in the Integration Build Plan.

It is critical that all system and subsystem interfaces be tested as well as system performance at this early stage.

The interfaces between the following subsystems will be tested:

1. Network statistics
2. CPU statistics
3. Memory statistics.

The most critical performance measures to test are:

1. Response time for Statistics to be displayed on the UI.
2. Response time to access the database.
3. Response time to access the database to generate the reports.

### **Testing Strategy**

1. The Test Strategy presents the recommended approach to the testing of the software applications.
2. The previous section on Test Requirements described *what* will be tested; this describes *how* it will be tested.
3. The main considerations for the test strategy are the techniques to be used and the criterion for knowing when the testing is completed.
4. In addition to the considerations provided for each test below, testing should only be executed using known, controlled databases, in the case of this project- SQLite.

### **Unit Testing**

* Unite Testing is performed by the members of the developing team.
* Simple tests are performed while the code is being developed which are narrow in scope.
* Tests are performed to check whether the code performs minimally.
* As the developers have a prior knowledge of how the code works, white box testing can be performed on the code.

### **System and Integration Testing**

The databases and the database processes should be tested as separate systems. These systems should be tested without the applications (as the interface to the data). Additional research into the DBMS needs to be performed to identify the tools / techniques that may exist to support the testing identified below.

Technique:

* The database access methods and processes testing will be done by passing valid and invalid data.
* To ensure that the data has been populated as intended in the database, testing should be done to ensure each events like data entered and data retrieved occurred properly.
* Testing should be done to ensure data Integrity.

### **Performance Testing**

Performance testing measures response times and other time sensitive requirements. The goal of Performance testing is to verify and validate the performance requirements have been achieved. Performance testing is usually executed several times.

Additionally, Performance tests can be used to profile and tune a system’s performance.

Technique:

* Modify data or modify scripts to increase/decrease the number of iterations.
* The goal is for successful completion of the scripts without any failures and within the expected / required time allocation.
* Performance testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurement.

### **Function Testing**

Function Testing should focus on any target requirements that can be traced directly to use cases (or business functions), and business rules. The goal is to test and verify data acceptance, processing, and retrieval and its appropriate implementation.

This type of testing is based upon black box techniques, that is, verifying the application by interacting with the application via the output interface and analyzing the output (results).

Technique:

* The expected results occur when valid data is used.
* The appropriate error / warning messages are displayed when invalid data is used.
* All identified defects should be addressed.

### **Configuration Testing**

Configuration testing verifies operation of the software on different software and hardware configurations. The goal is to achieve software functionality on any kind of Linux based operating system.

Technique:

* Check whether PC Applications are available, accessible on the target system.

### **Regression Testing**

* + Ensuring proper functionality of previous modules even after introducing new modules.
  + Usually performed at least once after adding new functionality.
  + By performing regression testing, user has confidence that the product is good until the previous version.

### **Hardware Requirements**

* Physical machines    : Desktop
* Operating systems    : Linux

### **Environment Requirements**

* Compilers            : GCC, JavaC
* Testing Environment : Junit
* IDE                : NetBeans
* Source code repository    : GitHub
* Build process        : Maven
* Database            : SQLite
* Instance             : Single

### **Test Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Author** | **Date** | **Status & Description** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### **Control Procedures**

### **Problem Reporting**

* If an incident is encountered during the testing process, those incidents are documented.

### **Features to Be Tested**

* Software features will be tested namely Network statistics, CPU statistics, and Memory statistics.

### **Schedules**

|  |  |  |
| --- | --- | --- |
| **Major Derivable** | **Author** | **Date** |
| Test Plan | All | 4/7/16 |
| Test Cases |  |  |
| Test Incident Reports |  |  |
| Test Summary Reports |  |  |
|  |  |  |

### **Risks/Assumptions**

Risks:

* Manual Testing is performed throughout the project (Automation testing is not performed).

Assumptions:

* All the team members are dedicating their time on completing their tasks.
* The design document may be modified based on the project requirements.

### **Approvals**

Name (In Capital Letters) Signature Date