**CS673S16 Software Engineering**

**Team X - Project Name**

**Tests Report**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Rohit Agrawal | Team Leader/ Implementation Leader | *Rohit Agrawal* | 10/16/2017 |
| Yashvardhan Nanavati | Backup Team Leader/ Configuration Leader/ Implementation Leader | *Yashvardhan Nanavati* | 10/16/2017 |
| Wenjun Shen | Requirement Leader/Environment and Integration Leader | *Wenjun Shen* | 10/17/2017 |
| Tianqi Xu | Requirement Leader/ Security Leader | *Tianqi Xu* | 10/17/2017 |
| Anirvan Maiti | Environment & Integration Leader  Design Leader | *Anirvan Maiti* | 10/18/2017 |
| Aswin Vasudevan | Implementation, Design and Security Leader | *Aswin Vasudevan* | 10/18/2017 |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1** | **Wenjun Shen** | **12/01/2017** | **Update test cases** |
| **2** | **Aswin Vasudevan** | **12/08/2017** | **Update testing** |

[Introduction](#_gjdgxs)

[Test Summary](#_30j0zll)

[Tests Reports](#_1fob9te)

[Testing Metrics](#_3znysh7)

[References](#_2et92p0)

[Glossary](#_tyjcwt)

# Introduction

In this section, give a summary of this tests report document.

The following report will conclude the types of testing, uses cases, and testing metrics we used in project Ukubuka following the test driven development guideline. We will cover three testing levels in the report: unit test, system test, and acceptance test. Each test is summarized step by step. We will also discuss the code coverage of our testing process.

# Test Summary

In this section, you will summarize what was tested and what happened, based on each test type.

* + Unit Tests
  + System Tests
  + Acceptance Tests

**System Tests**

The purpose of this test is to evaluate Ukubuka engine’s compliance with specific requirements. The Ukubuka engine takes in csv/json/xml files as inputs, and outputs csv/json/xml files. System tests are performed to ensure the Ukubuka engine executes schema correctly and transforms original files to expected files using the specific orders provided by schema.

1. First, We have to generate testing data. We partition a file into many files or merge many files into a file following certain rules.Partition or merge operation depends on which kind of transformation we want to test.
2. After we generate test data, we can pass it along with the scheme into Ukubuka engine and obtain the output files.
3. To compare expected files and actual files, we implement a script that can compare two csv/json/xml files and output whether they match or not.
4. We uses this script to compare the expected files and actual files, the result will be either pass or fail.

**Unit Tests:**Our project Ukubuka adheres to the concept of Test Driven Development so unit testing play a crucial role in the development phase through all the iterations. We have followed the norms that we write the unit test case initially and ensure that it fails then we develop the code just enough to satisfy the test cases. Our project is developed based on Spring framework. The bean invocation for the objects are dealt by the Spring container hence we have used Mockito and also Junit in the Unit testing.  
  
 **Mockito:**

Mockito is a java based mocking framework. Mockito is used to mock interfaces so that a dummy test related functionality can be added to a mock interface that can be used in unit testing. The Ukubuka have Autowired classes where we can inject the Mock to test the functionality rather than instantiating using new keyword anywhere in the project. This feature allows to test the functionality of a class in isolation. It is not computationally intense process as it neither uses a DB connection nor any other server. The Java reflection property is used in Mock object for any given interface. We have also added the behavior for the mock objects and verified it.

**Junit:**

We have also used Junit framework to test the class files along with the mockito framework which is based on Assertion. The Junit test is written to pass the assertion. The expected output is tested across the input and assertion is made to pass by writing the code after the unit test case. All the boundary conditions and exception behavior of the code is tested using this framework.

**Acceptance Tests:**The acceptance test is performed to determine whether the software system has met the requirement specification. In Ukubuka the key important requirement is to provide data related operations which was missing in the Kibana tool. We also state the other requirements in the initial iteration where we had elaborate analysis.   
The forms of Acceptance test used:  
1. User Acceptance Test

2. Business Acceptance Test

**User Acceptance Test:**  
The Acceptance criteria for this test are the usability, functional correctness and completeness. We tested the Ukubuka engine on real world datasets and ensured all functionalities are met. We also Beta tested the completed engine individually in all our machines.

**Business Acceptance Test:**

As we started the Ukubuka project as open source project, it was important for us to maintain up to date industrial practices. The project is available as generic solution to all kinds of visualization which was the key business point for our application that in future it can extended to technologies like php and Android. These business related points were incorporated our test plan and was ensured in the Integration and other testing phases.

# Tests Reports

In this section, you will give a detailed description of each test case performed and the result. You shall list what are existing tests developed in the previous semester and what are new tests developed currently.

For each test case, you can use the following template (or something based on the following template)

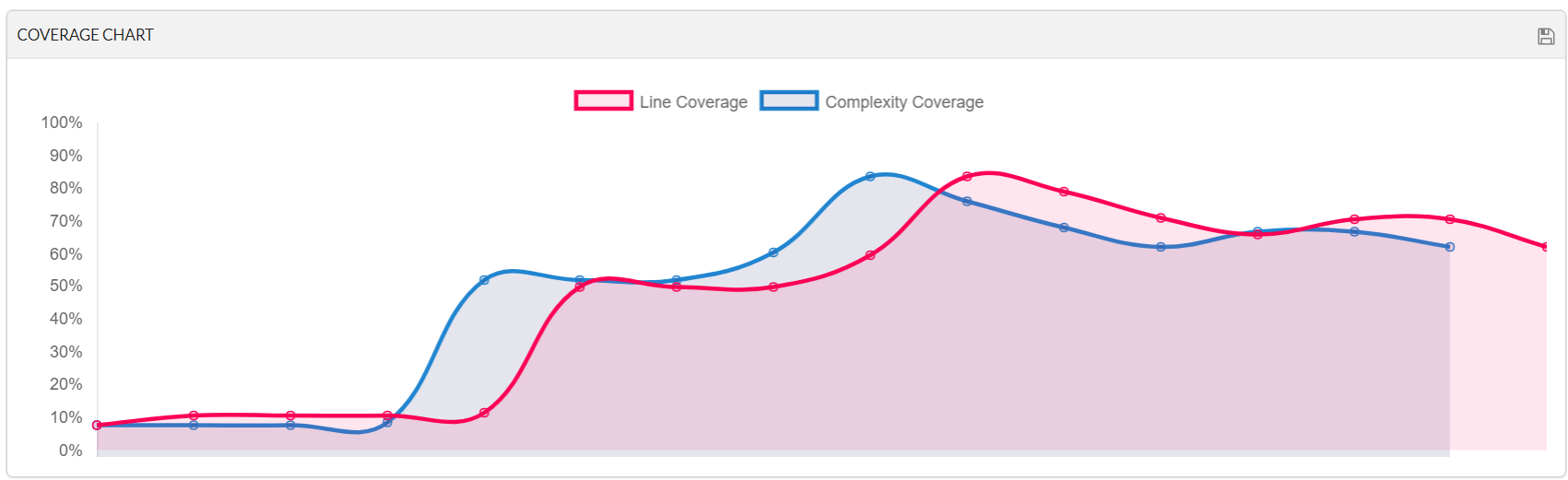
* Test case ID, name
* New or old:
* Test items: (what do you test )
* Test priority (high/medium/low)
* Dependencies (to other test case/requirement if any):
* Preconditions: (if any)
* input data:
* Test steps:
* Postconditions:
* Expected output:
* Actual output:
* Pass or Fail:
* Bug id/link: (this should link to your github issue id)
* Additional notes:

(You can use an additional table or document for this section)

The link to additional table of table:   
<https://docs.google.com/spreadsheets/d/1guwg5v_aPmacY9lOzPvw3lbdFA3q8_uxhiULe0S-Uhw/edit#gid=0>

# Testing Metrics

The code coverage has been integrated with our github repository build and we have used Eclemma for tracking the coverage on every forked project locally. Our project has code coverage of 89% and the chart below shows the line coverage and complexity coverage.



# References

# [1]: Code coverage: <http://www.eclemma.org/installation.html>

[2]: Mockito: <http://static.javadoc.io/org.mockito/mockito-core/2.13.0/org/mockito/Mockito.html>

[3]: J-unit: <http://junit.org/junit5/docs/current/api/>

[4]: Project: <https://github.com/ukubuka>

# 