Report on Software Testing (Gantt Project)

1 INTRODUCTION

This report is based on the software testing techniques used to test the web based application called GanttProject. It describes all the detailed analysis of the techniques used. The information is segregated into 4 steps/points. Ganttproject is a 100% pure Java application that lets you plan projects using Gantt charts. Ganttproject lets you easily break down a project into tasks, show dependencies, and manage resources. As it is written in Java, the Jar file runs on several operating systems, such as GNU/Linux, Microsoft Windows, MacOsX etc. Gantt Project as a whole is a huge project. It consist of more than 15000 LOC which is written by a combined effort of 26 developers for the past 10 years. Due to the short time for the final project we will be considering only the important features of the project like,

- Creating a Task.
- Creating a Resource.
- Creating Milestones.
- Creating a Gantt chart.
- Moving tasks and milestones.

The steps we will follow for the detailed analysis of the project will be strictly based on the checklist: -

TESTING: -

1) Setting up a schedule for the project and working on it accordingly.
2) Gathering the requirements necessary for the project.
3) Deciding the testing techniques which will be helpful for testing all the crucial aspects of our project.
4) Working on the test cases and gathering the results.
5) Result analysis stage.

QUALITY ASSURANCE: -

1) Setting the quality goals
2) Selecting a quality assurance activity.
3) Making adjustments based on feedback.

For our project we will use the **Capability Maturity Model (CMM) level 2** for effectively achieving the quality for the software.

Capability Maturity Model: -

'Capability Maturity Model', developed by the SEI. It's a model of 5 levels of organizational 'maturity' that determine effectiveness in delivering quality software. It is geared to large organizations such as large U.S. Defense Department contractors. However, many of the QA processes involved are appropriate to any organization, and if reasonably applied can be helpful.

Testing techniques that we are using for this project are: -

1) White Box testing: - To give an overview we will test all the internal functional aspects of our project. We will test each unit of our project individually (Unit testing). Also some part will be subjected for integration testing.
2) Usage based statistical testing: - The critical areas of the project will be tested based on the usage statistics. The usage statistics will be recovered by analyzing the areas from where the bug report/feedback is more.

2 RESULT ANALYSIS

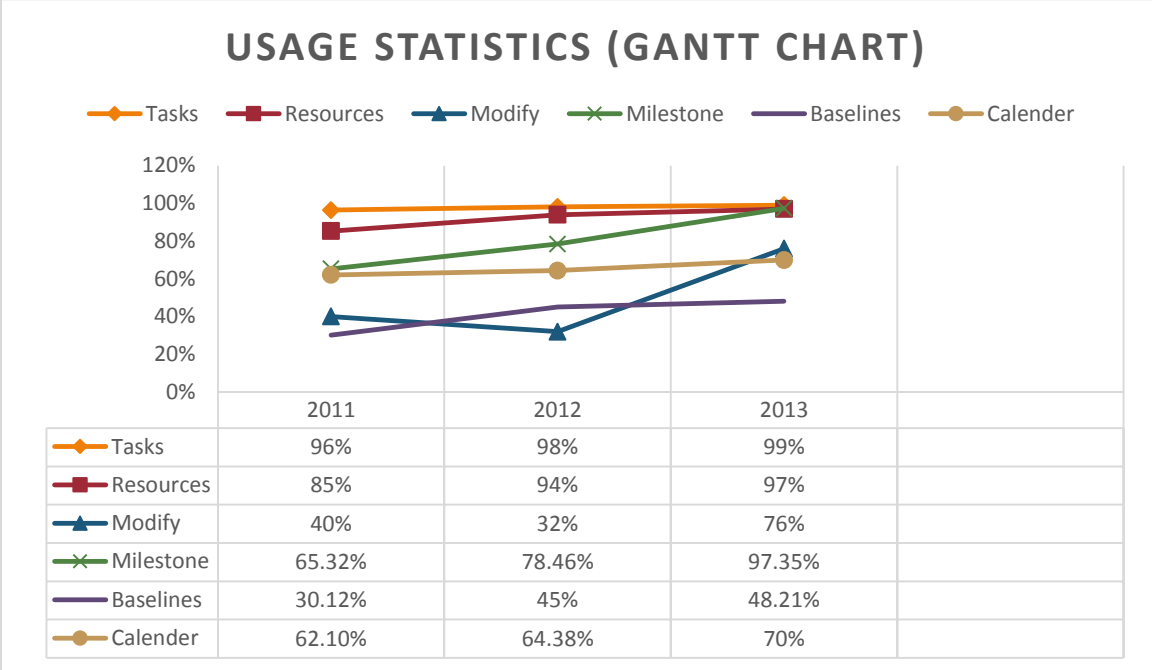
The Result Analysis of software testing analyzes the unit/module, subsystem integration, system, user acceptance, and security - as defined in the test plan. It records results of the tests, presents the capabilities and deficiencies for review, and provides a means of assessing software progression to the next stage of development or testing. Results of each type of test are added to the software development document for the module or system being tested.

The following section describes the results of each test performed.

1. Usage Based Statistical Testing: -

   One important testing technique, the usage-based statistical testing with Musa's operational profiles (OPs) shares the basic model with partition testing techniques, and enhances it to include probabilistic usage information. We next describe Musa 1 OP and their usage in testing.

   The usage statistics that we got from the users were as follows: -



	2011	2012	2013	
Tasks	96%	98%	99%	
Resources	85%	94%	97%	
Modify	40%	32%	76%	
Milestone	65.32%	78.46%	97.35%	
Baselines	30.12%	45%	48.21%	
Calender	62.10%	64.38%	70%	

The above statistics show that most of the usage is related to the tasks, resources, milestones of the Gantt chart. Therefore we will apply White Box Testing in this critical areas.

2. White Box Testing: -

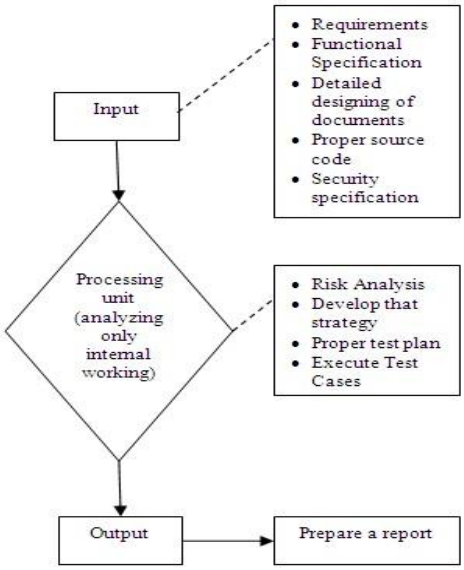
Step 1: Input ~~includes~~ → – Requirement.

– Functional specification.
– Detailed designing of documents.
– Proper source code.
– Security specifications are must.

Step 2: Processing Unit (Analyzing internal working only)

- Perform risk analysis to guide whole testing process
- Proper test plan
- Execute test cases and communicate results

Step 3: Output → Prepare final report



2.1 LOOP TESTING: -

Loop testing is another type of white box testing which exclusively focuses on the validity of loop construct. Loops are simple to test unless dependencies exist between the loops or among the loop and the code it contain. There are four classes of loops:

a) Simple Loop

a) Nested Loop

b) Simple Loop: -

In this we test the simple if else and for loops in our code. All the IF-ELSE loops pass the following steps: -

Step 1: Skip the loop entirely

Step 2: Only one pass through the loop

Step 3: Two passes through the loop

Step 4: m passes through the loop where n>m

Step 5: n-1, n, n+1 passes through the loops

Where n it's a maximum no. of allowable passes through loops.

c) Nested Loop: -

In this we test the nested if else if loops in our code. All the IF-ELSE-IF loops pass the following steps: -

Step 1: Set all the other loops to minimum value and start at the innermost loop

Step 2: Conduct simple loop test for the innermost loop and holding the outer loops at their minimum iteration parameter value

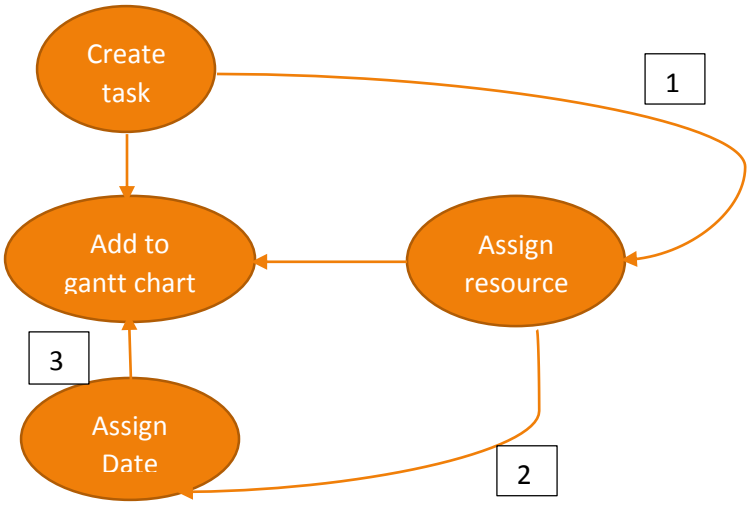
Step 3: Performing test for the next loop and work outward

Step 4: Continue until all the loops have been tested

2.2 Data Flow Testing

Data flow testing is another type of white box testing which looks at how data moves within a program. In data flow testing the control flow graph is annoted with the information about how the program variables are defined and used.

Following is the sample result for a data flow testing: -



2.3 TEST CASES

By performing the above tests on the codes the following defects were discovered: -

Total number of defects found: - 33

Total number of test cases: - 115

Following are few sample test cases from our project and the defects found respectively: -

Test case #57

Definition: -

Change date of a sample project/ Resize start date to fit data.

Expected Result: -

Date Column Resizes.

Actual Result: -

ERROR!! Size remains the same.

Solution: -

Modify /ganttproject/src/net/sourceforge/ganttproject/GPTreeTableBase.java

```
...
776  776        });
777  777        }
778  778        {
779       -     result.add(new GPAction("columns.pack.label") {
780       -     @Override
781       -     public void actionPerformed(ActionEvent arg0) {
782       -     }
783       -     });
784  779        GPAction fitAction = new GPAction("columns.fit.label") {
785  780        @Override
786  781        public void actionPerformed(ActionEvent e) {
...
849  844        int width = comp.getPreferredSize().width;
850  845
851  846        // Get maximum width of column data
852       -     renderer = tableColumn.getCellRenderer();
     847  +
853  848        for (int r = 0; r < getTable().getRowCount(); r++) {
```

```
            849  +  renderer = table.getCellRenderer(r, column.getOrder());
854         850     comp = renderer.getTableCellRendererComponent(
855         851     table, table.getValueAt(r, column.getOrder()), false, false, r, column.getOrder());
856         852     width = Math.max(width, comp.getPreferredSize().width);
...
```

Test case #23

Definition: -

Final date always incremented by 1 for example if we save start date as 14th December and end date as 15th December. Save and re-open again.

Expected result: -

The entered date should remain same.

Actual result: -

End date changes to 16th December.

Solution: -

Modify /ganttproject/src/net/sourceforge/ganttproject/gui/DateIntervalListEditor.jav

```
...
52  52      private final Date myVisibleEnd;
53  53      private final Date myModelEnd;
54  54
55      -   public DateInterval(Date start, Date end) {
        55  + private DateInterval(Date start, Date visibleEnd, Date modelEnd) {
56  56      this.start = start;
57      -   myVisibleEnd = end;
58      -   myModelEnd = GPTimeUnitStack.DAY.adjustRight(end);
        57  + myVisibleEnd = visibleEnd;
        58  + myModelEnd = modelEnd;
59  59      }
60  60      @Override
61  61      public boolean equals(Object obj) {
```

```
...
76  76      return myModelEnd;
77  77      }
78  78
    79  +   public static DateInterval createFromModelDates(Date start, Date end) {
    80  +   return new DateInterval(start, GPTimeUnitStack.DAY.adjustLeft(GPTimeUnitStack.DAY.jumpLeft(end)), end);
    81  +   }
    82  +   public static DateInterval createFromVisibleDates(Date start, Date end) {
    83  +   return new DateInterval(start, end, GPTimeUnitStack.DAY.adjustRight(end));
    84  +   }
79  85      }
80  86      public static interface DateIntervalModel {
81  87      DateInterval[] getIntervals();
...
158 164     myAddAction = new GPAction("add") {
159 165     @Override
160 166     public void actionPerformed(ActionEvent e) {
161     -   myIntervalsModel.add(new DateInterval(myStart.getValue(), myFinish.getValue()));
    167 +   myIntervalsModel.add(DateInterval.createFromVisibleDates(myStart.getValue(), myFinish.getValue()));
162 168     myListModel.update();
163 169     }
164 170     };
...
```

Modify /ganttproject/src/net/sourceforge/ganttproject/gui/GanttDialogPerson.java

```
...
241 241     DefaultListModel daysOff = person.getDaysOff();
242 242     for (int i=0; i<daysOff.getSize(); i++) {
243 243     GanttDaysOff next = (GanttDaysOff) daysOff.get(i);
```

```
244        -  myDaysOffModel.add(new DateIntervalListEditor.DateInterval(next.getStart().getTime(),
              next.getFinish().getTime()));
       244 +  myDaysOffModel.add(DateIntervalListEditor.DateInterval.createFromModelDates(next.getStart().getTim
              e(), next.getFinish().getTime()));
245    245       }
246    246
              DateIntervalListEditor editor = new DateIntervalListEditor(myDaysOffModel);
247    247
              return editor;
...
```

Modify/ganttproject/src/net/sourceforge/ganttproject/gui/GanttDialogPublicHoliday.java

```
...
39     39        public GanttDialogPublicHoliday(IGanttProject project) {
40     40        publicHolidays = new DateIntervalListEditor.DefaultDateIntervalModel();
41     41        for (Date d : project.getActiveCalendar().getPublicHolidays()) {
42         -     publicHolidays.add(new DateIntervalListEditor.DateInterval(d,d));
           42 +  publicHolidays.add(DateIntervalListEditor.DateInterval.createFromVisibleDates(d,d));
43     43        }
44     44
45     45        publicHolidayBean = new DateIntervalListEditor(publicHolidays);
...
```

2.2.1 Statement Coverage

Statement coverage is a measure of the percentage of statements that have been executed by test cases. Anything less than 100% statement coverage means that not all lines of code have been executed. We can achieve statement coverage by identifying cyclomatic number an executing this minimum set of test cases. Measure benefit of statement coverage is that it is greatly able to isolate the portion of code, which could not be executed. Since it tends to become expensive, the developer chose a better testing technique called branch coverage or decision coverage.

2.2.2 Branch Coverage

A stronger logic coverage criterion is known as branch coverage or decision coverage. Branch coverage or decision coverage is a measure of the percentage of the decision point of the program have been evaluated as both true and false in test cases. Examples of branch coverage-DO statements, IF statements and multiway GOTO statements. Branch coverage is usually shown to satisfy statement coverage. By 100% branch coverage we mean that every control flow graph is traversed.

2.2.3 Condition Coverage

A criterion which is stronger than decision coverage is condition coverage. It is a measure of percentage of Boolean sub-expressions of the program that have been evaluated as both true and false outcome in test cases.

// CMM level 2 diagrams will come here.

3 RESULT ANALYSIS AND FOLLOW UP ACTIONS

The two techniques we did above were UBST and WBT. Usage based statistical testing was useful in finding the critical areas affected with most number of defects. By using the information from UBST we were able to perform WBT successfully in those critical areas. WBT was the detailed and in depth testing of the critical modules, like testing the internal code of the functions. The WBT included testing techniques like Loop Testing, Branch Testing etc. as mentioned above. Also the above coverage techniques were used to see to it that all the codes in the critical area was covered successfully (because we were paranoid.)

3.1 EDUCATIONAL BENEFITS

There was a lot of benefits that we were able to grasp from this project. Again a special thanks to Dr. Jeff Tian for giving a project that was not just a project for the purpose of getting a grade, but also a project that will be very beneficial in the future. To list a few of the benefits: -

1) The major testing techniques (like we used UBST and WBT), their implementation, how they work in the actual testing environment.

2) The actual environment of doing a group project like strictly following the schedule and submitting the work on time (thanks to the milestones). Doing the specified work assigned to the group member in time. It was a great exposure to that kind of environment.
3) Also the workflow that should be followed while performing the testing and QA activities. We learned that in order to perform a successful testing the GOAL for the testing should be clear in the first place. Also the requirements for performing the testing is all very important.
4) We came across a lot of errors while performing the testing and also learned about how to overcome such similar errors in the future while performing WBT or UBST.
5) The main thing that we can conclude after performing the testing is that, Simplicity is the key to perform an effective testing. The simpler the test cases the simple will be the work flow.

3.2 MAJOR FOLLOW UP ACTIONS

The best way to improve the software or remove the defects after product release is through user feedback. Therefore there is a forum for the user who use Gantt in their day to day life. The forum mainly focus on the tricks to use the software, also the users can report if they have encountered any defect so that the developers can rectify those defects in their next release.