Beets

Test Plan

12/13/2012

TABLE OF CONTENTS

[1 Introduction](#h.frxumwlfs7cx)

[1.1 Purpose of the Test Plan Document](#h.ke9gbedj0ff)

[1.2 Audience of the Test Plan Document](#h.kixe9dfpmzgv)

[1.3 Project Description](#h.350zqdyv9kz5)

[1.3.1 System Design](#h.r1b62meho47e)

[1.3.2 Current State of the Project](#h.jpuquvwxcn8c)

[1.4 Technologies](#h.yfrgz24z3k9j)

[1.4.1 System Technologies](#h.5e25djyzh2qq)

[1.4.2 Testing Environment](#h.6p7i1byldd2y)

[2 User Acceptance Testing](#h.y6flcnxrc0zi)

[2.1 Risks / Issues](#h.hgxsluwxowds)

[2.2 Environmental / Training Needs](#h.1ljsd9k)

[2.3 Items to be Tested / Not Tested](#h.4du1wux)

[2.4 Test Approach](#h.2szc72q)

[2.5 Entry / Exit Criteria](#h.279ka65)

[2.6 Pass / Fail Criteria](#h.3s49zyc)

[2.7 Deliverables](#h.meukdy)

[3 Load & Stress Testing](#h.bsrzs36gxhcd)

[3.1 Risks / Issues](#h.h5mdpsc691kw)

[3.2 Environmental / Training Needs](#h.stmg1axzknl7)

[3.3 Items to be Tested / Not Tested](#h.4sse0ab2j6e3)

[3.4 Test Approach](#h.oi9gzlgturbk)

[3.5 Entry / Exit Criteria](#h.r9m2f6sxht78)

[3.6 Pass / Fail Criteria](#h.254tvdf2imls)

[3.7 Deliverables](#h.ui3oj8951dmz)

[4 Compatibility Testing](#h.ok6cpm3evvsz)

[4.1 Risks / Issues](#h.n5apwl63ll1q)

[4.2 Environmental / Training Needs](#h.g95d8l6mzcrm)

[4.3 Items to be Tested / Not Tested](#h.me2vzuowfku7)

[4.4 Test Approach](#h.6jex23yamcyb)

[4.5 Entry / Exit Criteria](#h.9x5dzdnp3rkg)

[4.6 Pass / Fail Criteria](#h.62cmpknysejn)

[4.7 Deliverables](#h.c75u0pctryvf)

[5 Mutation & Coverage Testing](#h.basz4kc4yho0)

[5.1 Risks / Issues](#h.8ujrt8ufsumq)

[5.2 Environmental / Training Needs](#h.hwbenuqlc1pr)

[5.3 Items to be Tested / Not Tested](#h.s0ieyw3mz8h)

[5.4 Test Approach](#h.lt343x80672z)

[5.5 Entry / Exit Criteria](#h.ylc6dw4j6exq)

[5.6 Pass / Fail Criteria](#h.l70x6yaugep7)

[5.7 Deliverables](#h.2nsq2eu05g2u)

[6 Unit Testing](#h.8kyewhppkhtz)

[6.1 Risks / Issues](#h.mb8rczigwo74)

[6.2 Environmental / Training Needs](#h.2uofsrupe196)

[6.3 Items to be Tested / Not Tested](#h.pcecvg1bylaz)

[6.4 Test Approach](#h.9f62zo3udyu2)

[6.5 Entry / Exit Criteria](#h.wllc0xqfitz)

[6.6 Pass / Fail Criteria](#h.6gzk28bpy7po)

[6.7 Deliverables](#h.653em594dthc)

[7 Code Inspections](#h.5nxmhiyknmd)

[7.1 Risks / Issues](#h.b4l4v1lhb9b8)

[7.2 Environmental / Training Needs](#h.mdyg3tmqbxmt)

[7.3 Items to be Tested / Not Tested](#h.nsyrnhq97uyg)

[7.4 Test Approach](#h.kldayhon7m9y)

[7.5 Entry / Exit Criteria](#h.4cz4di69nna3)

[7.6 Pass / Fail Criteria](#h.5xx8umtsooxf)

[7.7 Deliverables](#h.waclex8f1z65)

[Appendix A: References](#h.8gvfxq7p5l1r)

[Appendix B: Key Terms](#h.yrxdm37lm3k2)

# 

# 

# 1 Introduction

## 1.1 Purpose of the Test Plan Document

This document states and tracks the appropriate information for effectively and thoroughly testing the open source Beets software . The goal of this test plan is to provide an effective means of communication between the developers and the quality assurance teams in regards to the testing schedule, scope, resources, and strategies. This document will also define quality and reliability goals for the Beets software in order to measure the amount of testing success that was achieved at the end of the testing phase. The test plan will define which areas of the software system will be tested, which areas will not be tested, and justification for both.

## 1.2 Audience of the Test Plan Document

Affected audience of this Test Plan document include:

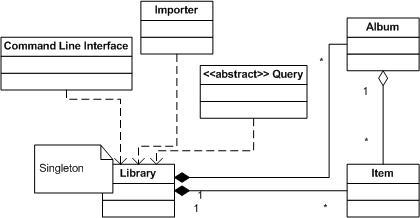
* Beets Developers
* Quality Assurance Team
* Users of the Beets Software

## 1.3 Project Description

Beets is a music collection organizer which allows the user to catalog one’s music collection by improving areas such as the metadata of the music track by using the MusicBrainz database. It is a product that is meant to be used by experienced computer users comfortable with interfacing the Beets program with the command line of any operating system. In addition, some of the big features of the Beets program is the ability to embed or extract any album arts and the ability to view one’s music library through a web browser and play it on the browser itself. Beets is a powerful but simple to program to organize one’s music efficiently and quickly.

### 1.3.1 System Design

Basic system design:



*Library*

Singleton object responsible for adding, loading, storing, removing and moving items

*Item*

Object to represent a song or track.

*Album*

Object to represent a collection of items.

*Query*

Abstract base class for defining the criterion that matches items or albums in the database. Each subclass extending Query must implement the clause() and match(item) methods.

*Importer*

Object responsible for adding music to the library by finding it in the local filesystem.

*Command Line Interface*

Module for holding interactions with the user through the terminal.

### 1.3.2 Current State of the Project

The Beets project could be found at <http://beets.radbox.org/>.

The latest release of the project is a candidate release version *1.0rc1* releasedon 12/17/2012 (<http://beets.readthedocs.org/en/1.0rc1/changelog.html>) with numerous plugins. This release version is feature complete and currently the only activity is bug fixing prior to the official release. The main developer and customer identified for the Beets project is Adrian Sampson ([adrian@radbox.org](mailto:adrian@radbox.org)).

The Beets project currently has 26 active developers and/or committers. The most recent commit to the open-source project was dated at 12/19/2012 (as of 12/20). Many unit tests exist, stated to have 60% coverage, but certain areas have fallen behind. As of 12/20/2012 the Beets projects currently has 92 reported bugs majority with a medium priority, low priority following, and high priority bugs being the least.

Active Committers (<https://github.com/sampsyo/beets/graphs/contributors>)

Commits (<https://github.com/sampsyo/beets/graphs/commit-activity>)

Code Frequency (<https://github.com/sampsyo/beets/graphs/code-frequency>)

Punch Card (<https://github.com/sampsyo/beets/graphs/punch-card>)

Bugs (<http://code.google.com/p/beets/issues/list>)

Testing Comments (<https://github.com/sampsyo/beets/wiki/Testing>)

Beets currently has no formal requirement documentation.

## 1.4 Technologies

### 1.4.1 System Technologies

Beets can be deployed on Mac OS X, Windows, and Linux machines. Beets is slolely written in Python (<http://www.python.org/>), compatible with Python version 2.6 and above.

Beets currently uses Python nose (<https://nose.readthedocs.org/en/latest/>) for unit testing and coverage.py (<http://nedbatchelder.com/code/coverage/>) for coverage testing. Beets uses google code issue tracking (<http://code.google.com/p/beets/issues/list>) for bug tracking.

Beets depends on MusicBrainz (<http://musicbrainz.org/>) as the main library from which album, artist, track, etc information is imported from. MusicBrainz is an open music encyclopedia that collects music metadata and makes it available to the public.

Beets utilizes Sqlite (<http://www.sqlite.org/>) for its library in order to create, read, update and delete the metadata associated with each track.

Beets uses Mercurial (<http://mercurial.selenic.com/>) for Version Control. Check out is found here <http://code.google.com/p/beets/source/checkout>.

Beets uses several wiki sites:

<https://github.com/sampsyo/beets/wiki>

<http://code.google.com/p/beets/>

<http://beets.readthedocs.org/en/1.0rc1/changelog.html>

### 1.4.2 Testing Environment

The Integrated Development Environment being used to test Beets is Eclipse (<http://www.eclipse.org/>). The Pydev eclipse plugin (<http://pydev.org/>) will is used for compiling the source files of the software. Testing frameworks that will be used are Python nose, currently used by Beets.

# 

# 

# 2 User Acceptance Testing

## 2.1 Risks / Issues

As acceptance testing is typically the last testing strategy, the order of the testing process is of most concern. The order in which testing phases will be completed will depend on the schedule of other testers. There may be multiple test phases occurring simultaneously.

## 2.2 Environmental / Training Needs

Acceptance testing is considered a black box testing approach. Testing will need to be carried out in an environment that is identical to the real-world and from the customer’s point of view. Since this testing phase tests the functionality, opposed to the technicality of the product, an independent test team can be used to go through the test cases. Those testers would require basic training to understand the purpose of the testing.

## 2.3 Items to be Tested / Not Tested

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to Test** | **Test Description** | **Test Date** | **Responsibility** |
| **Loading Data** | **Data (music files) will be loaded into the system.** | **12.24.12**  **to**  **01.07.13** | **Jenny** |
| **Cleaning Metadata** | **Follow the steps in the product documentation to use the feature to clean up metadata.** | **12.24.12**  **to**  **01.07.13** | **Jenny** |
| **Library Management** | **Follow the steps in the product documentation to use the feature to manage the music library.** | **12.24.12**  **to**  **01.07.13** | **Jenny** |
| **Music Playback** | **Follow the steps in the product documentation to use the feature to playback music using the GUI interface.** | **12.24.12**  **to**  **01.07.13** | **Jenny** |

## 2.4 Test Approach

The purpose of acceptance testing is to allow end users to test the program before accepting the product as complete. Thus, acceptance testing is typically the final step in the development cycle after other testing strategies: unit testing, integration testing, and system testing, have been completed. To ensure that this testing phase is effective to the customer, a wide range of test cases will be provided to test against the product’s requirements/specifications. Those test cases will be written based on use cases defined during the requirements stage. For each test case, the steps required to be taken have to be precise.

## 2.5 Entry / Exit Criteria

Acceptance testing typically begins as soon as unit testing, integration testing, and system testing have been completed. Due to the undetermined nature of this project, the order may not be followed as strictly. Testing may begin as soon as all the test cases have been written and finalized. Testing will end once all test cases have been gone through by at least two individuals or groups of people.

## 2.6 Pass / Fail Criteria

A test case will pass if it meets all the criteria listed in the expected outcome. If the actual results do not match completely and is a minor issue, i.e. spelling error, it may be marked as neutral. Otherwise, the test case will fail if the criteria is not implemented, or is implemented incorrectly. Explanations will be needed for neutral and failed test cases.

## 2.7 Deliverables

The user acceptance test document will contain all of the test cases with the associated user story, preconditions, description in steps, expected outcome, and actual outcome (pass/fail/neutral). Any bugs encountered may be expanded to include the steps to replicate the bug in a separate section of the document.

# 

# 

# 3 Load & Stress Testing

## 3.1 Risks / Issues

Both load and stress testing require high amount of resources that require time, money, and effort to obtain. Planning the tests would require understanding of product requirements and the goal of the load and stress testing process.

## 3.2 Environmental / Training Needs

The environment used during the testing process must be a controlled environment since resources have to be closely monitored and precise. Testing would require testers to understand how to examine the entirety of the environment that the program will be run on. The testers would need to know how to monitor resources consumed and measure system response time.

## 3.3 Items to be Tested / Not Tested

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to Test** | **Test Description** | **Test Date** | **Responsibility** |
| **Peak Operation**  **(load)** | **Large quantities of data (music files) will be queued. Process will be repeated using various load levels to obtain upper threshold of operation.**  **(Tests will be repeated.)** | **12.24.12**  **to**  **01.31.13** | **Eric Caron** |
| **System Failure**  **(stress)** | **Large quantities of data (music files) will be queued. Process will be repeated until system failure.**  **(Tests will be repeated.)** | **12.24.12**  **to**  **01.31.13** | **Shun Mok Bhark** |
| **Resource**  **Consumption**  **(load)** | **Multiple instances of the program will be used with varying amounts of data. Resources will monitor response times for each system.** | **12.24.12**  **to**  **01.31.13** | **Eric Caron** |

## 3.4 Test Approach

Both stress and load testing require approaches that test the program under abnormal circumstances, rather than testing when the program would behave in what could be considered “correctly” under normal program use and behavior.

Stress testing would require close monitoring of consumed resources, including CPU, internal memory, and external disk space. The amount of resources will be recorded as more data is fed into the system. This process will be repeated until the system results in a failure. The recovery will be analyzed and recorded.

Load testing could be approached by opening multiple instances of the program and feeding large amounts of data into all of them. The time for each of the system to respond will be recorded. Additionally, another instance of the program may be opened with a smaller set of data and the reaction time for that instance could be compared to the other instances that demanded more resources. The goal of load testing will be to obtain the upper threshold for which the program is still operational.

## 3.5 Entry / Exit Criteria

Testing for stress and load can both occur towards the end of the development cycle. They would both require repeated runs of the tests to ensure precision in resources and recovery data gathered. Once the results prove to be consistent, testing can be concluded.

## 3.6 Pass / Fail Criteria

For stress testing, the tests would be considered as passed if the system could recover gracefully and could be restored back to the previous state (may/may not be possible). The test would fail if the system failed and could not recover, assuming recovery features were implemented. Otherwise, the test would be marked neutral, with the test case containing the largest amount of data marked as the most relevant test case (each test case is run multiple times).

For load testing, the tests would pass if no bugs, including memory management bugs, memory leaks, buffer overflows, etc. were uncovered despite running the program at above normal or peak performance. The tests would fail otherwise.

## 3.7 Deliverables

Detailed reports with the amount of data and resources used, including charts to provide a better visual of the results. The results of the recovery would also be described in the stress testing report. Any bugs found would also be described with the steps taken that produced that bug.

# 

# 

# 4 Compatibility Testing

## 4.1 Risks / Issues

Compatibility testing requires access and familiarity with the variety of operating systems out there in the market. This could include testing all editions of WIndows, MacOSX, and Linux. By doing market research ahead of time its possible to figure out what are the most popular operating systems and target the compatibility testing to those specific operating systems.

## 4.2 Environmental / Training Needs

For this test a variety of operating systems will be needed. Those would include the systems found to be popular when research was performed. Those operating system would include: Windows (XP and above), Mac OS X (10.5 and above), and Linux (Redhat and Ubuntu). Proficiency with the operating system as well as your systems command line interface is also needed. Knowledge of software installation will also needed to install the application. That information can be found at <http://beets.readthedocs.org/en/1.0b15/guides/main.html>.

## 4.3 Items to be Tested / Not Tested

The items to be tested include the successful install and use of beets on the given system.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to Test** | **Test Description** | **Test Date** | **Responsibility** |
| **Windows Install**  **(XP and above)** | **Check to see for successful Installation and ability to run** | **12/24/12 - 1/07/13** | **Chris N** |
| **Mac OS X Install**  **(10.5 and above)** | **Check to see for successful Installation and ability to run** | **12/24/12 - 1/07/13** | **Chris N** |
| **Linux Install**  **(Red hat and Ubuntu)** | **Check to see for successful Installation and ability to run** | **12/24/12 - 1/07/13** | **Chris N** |

## 4.4 Test Approach

Since compatibility testing requires a working knowledge of WIndows, MacOSX, and Linux, It is suggested that installation of software onto those operating systems are known. Once that information is known, testing can begin.

For each of the tests installation instructions are needed. Those can be found at <http://beets.readthedocs.org/en/latest/guides/main.html>. Once the installation instructions are understood, and attempt at installation will be performed on each operating system.

## 4.5 Entry / Exit Criteria

In order for testing to begin you must a a stable version of the software. The software must also be packaged so installation is possible. Testing is complete when that version is able to successfully install and run on the targeted operating system.

## 4.6 Pass / Fail Criteria

For a successful test the software must be able to install and run on the operating system. If the software is unable to install or run that would be considered a failure.

## 4.7 Deliverables

A spreadsheet on the installation outcome for each of the operating systems being tested. This spreadsheet would include whether the software was able to be installed onto the system successfully or not. Also on the spreadsheet would be an average time on how long the installation process took. The time is important for refining the installation process for the future.

# 

# 

# 5 Mutation & Coverage Testing

## 5.1 Risks / Issues

With several versions of python and several testing frameworks, setting up a stable testing environment may not be a straightforward process. In the event that the tester cannot run the tests, the team will work together to set up the environment, and the developer may be contacted for information about the development environment.

## 5.2 Environmental / Training Needs

Coverage testing will use coverage.py. Mutation testing will require software such as PyMuTester or MutPy. After these are installed, execution is simple, but the tester may need a small amount of time to analyze their output to discern how it is useful.

## 5.3 Items to be Tested / Not Tested

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to Test** | **Test Description** | **Test Date** | **Responsibility** |
| **Existing unit tests** | **Determine the current coverage and strength of the test suite** | **12/24/12 - 1/1/13** | **David** |
| **Additional unit tests** | **Determine the coverage and strength of the test suite when augmented by this team** | **1/1/13 - 1/14/13** | **David** |

## 5.4 Test Approach

Coverage and Mutation tests will be run on all existing unit tests and the results documented to reveal problem areas in the tests in addition to the product. These tests will be run again after extra unit tests have been added and existing ones modified.

## 5.5 Entry / Exit Criteria

Testing of the current project can begin immediately. These initial tests are finished when they produce quantifications of the current coverage of the test suite and how many mutants are killed (and which ones remain). Then, testing will resume using an updated test suite once all new unit tests have been implemented. The exit criteria are the same as the first time.

## 5.6 Pass / Fail Criteria

A test item fails the Coverage test if and only if it is not exercised within unit tests. An item fails the Mutation test if and only if unit tests fail to “kill” a “mutant” (the program doesn’t crash and no unit tests fail when a small error is injected into the code).

## 5.7 Deliverables

Coverage tests will result in summary charts describing how much code is covered by unit tests. It will also reveal exactly which code hasn’t been covered, though it may be impractical to document this detailed information. Mutation tests will result in summary charts describing the proportion of mutants killed by unit tests. It will also list each “live” mutant, and this list will be documented with the intention of specifically targeting and killing these mutants.

# 6 Unit Testing

## 6.1 Risks / Issues

As the primary code of the Beets is updating frequently there will be issues in creating new unit tests as our local repository may not be updated as frequently. In addition, code that existed before and which was included in the unit test may become obsolete further failing any unit tests involved. To mitigate this issues, talks with the developers will need to be scheduled for notification on any upcoming release of the beets system.

## 6.2 Environmental / Training Needs

A Linux VM with Python installed will be required to run the program and any unit tests. Creating unit tests in Python will be required and will be using the unittest (PyUnit) package. A music library will be required to run some unit tests requiring the use of a music track or library to perform the tests. Creating unit tests will require the Eclipse IDE with the PyDev plugin installed.

## 6.3 Items to be Tested / Not Tested

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to Test** | **Test Description** | **Test Date** | **Responsibility** |
| Existing Unit Tests | Existing Unit Tests will be re-run to verify that it still passes or fails. If the tests fails then additional analysis will need to be performed to see if any refactoring will be required.  Once the refactoring is complete, it will be verified by the lead developer. | 1/1/2013-1/30/2013 | Shun Mok Bhark |
| New Unit Tests | New Unit Tests will be added if there are any missing unit tests. In addition, if there are any areas of the product code that lacks unit testing, new tests will be created to address the missing areas.  Any new unit tests created will be verified by the lead developer of the product to make sure that the tests the appropriate features or requirements. | 1/1/2013-1/30/2013 | Shun Mok Bhark |

## 

## 6.4 Test Approach

Currently, there are several unit tests which have already been coded by the developer of beets to test the system. The existing unit tests will need to be run again to make sure that it still passes and if there are any failures the unit tests be will be fixed to make it pass. If there are any failures in existing unit tests, a thorough analysis will need to be completed to see if it is due to the test code being outdated or if any new features were added to the product to fail the test execution.

After the existing test have been verified passing new unit tests will be created to fill in any missing unit tests or to test newly introduced features. The creation of the any new unit tests will require input from the developer to determine if it is necessary and tests the appropriate features.

## 6.5 Entry / Exit Criteria

The entry criteria for unit testing are:

* The system is up-to-date and stable.
* The system is fully complete with no major issues that may cause errors in testing.
* There is no major upcoming release that may hinder the unit test.
* Technical documentation of product is complete.

The exit criteria for unit testing are:

* All Unit Tests have passed.
* Documentation of each unit test is complete.
* Units Tests are well commented in code to provide ease of reading for future developers or testers.
* Any unused code or unused or unnecessary unit tests are removed.
* Unit Test are complete with respect to testing the requirements or features that are present in the requirements document or from a list provided by the developer(s).
* Any and all bugs found during testing are fixed.
* No critical or major bugs are found and left in the system.

## 6.6 Pass / Fail Criteria

A test case will be marked as pass if it matches the correct and expected output or value of the feature being tested. If it does not produce the correct output or value then the test case will be marked as a failure.

## 6.7 Deliverables

An overall spreadsheet will be created to document all unit tests with its description and purpose. Any failures and passes that occurred during unit testing will be recorded. All failed unit tests will have a section on what was done to resolve the failure to provide detailed documentation. A coverage report will also be provided how much coverage the unit testing covered actual product code.

# 7 Code Inspections

## 7.1 Risks / Issues

Code inspections require a minimum of 3-4 members of the team present to conduct the code reviews. The moderator has to work with the team to decide which classes are important enough to dedicate time and energy to conduct the reviews. The members participating in the reviews must have a decent amount of programming knowledge to understand parts of the code, including the logic behind it.

## 7.2 Environmental / Training Needs

The moderator has to ensure that all of the team members participating in the code review process knows which classes will be analyzed. Every team member has to have the information necessary to annotate and understand the code being reviewed prior to the date of the meeting. Members have to understand that the goal of the code inspections is not to find solutions to the bugs, but to find the bugs themselves.

## 7.3 Items to be Tested / Not Tested

|  |  |
| --- | --- |
| **Item to Test** | **Test Date** |
| beets/autotag/\_\_init\_\_.py | February 5, 2013 |
| beets/autotag/hooks.py | February 5, 2013 |
| beets/autotag/match.py | February 5, 2013 |
| beets/autotag/mb.py | February 5, 2013 |

|  |  |
| --- | --- |
| **Role** | **Name** |
| Moderator | Jenny Zhen |
| Scribe | Eric Caron |
| Reader | Chris Ketant |
| Tester | David Wilson |

## 

## 7.4 Test Approach

The moderator will distribute information needed for the members to prepare for the code inspections on the designated date for each class being tested. The expectations for what needs to be done to prepare for said inspections will be explained to each team member. Prior to the meeting, the members will read through the code and annotate any parts of the code that the members believes to be worthy of discussion. On the day of the code inspections, the moderator and the reader will lead the team through the code, section by section. Any bugs found will be discussed briefly to determine the severity of the bug before the scribe takes note of the class the code was in, the line numbers, and a brief description of the bug. After code inspections have been conducted for all bugs, the moderator will complete the task by logging all bugs into the bug tracker, FreeBugBase.

## 7.5 Entry / Exit Criteria

Testing begins after all members participating in code inspections have prepared for the reviews by annotating and understanding the code. All members should have thoughts and concerns brought up on the day of the meeting regarding anything they believe is a bug in the code.

Testing is complete after all members have gone through all of the classes required to be reviewed, and all problems from each member has been briefly discussed. Additionally, all bugs and their severity have been logged into the bug tracking system for further analysis by the Beets development and quality assurance team.

## 7.6 Pass / Fail Criteria

Anything that does not meet the requirements of what the code was intended to do is a bug, or a defect. These bugs are categorized by their severity, which may range from low, medium, to high. Each bug is also assigned a priority, but that is highly based on their severity. A bug is considered low severity if it is a typo or style issue. A bug that has a high possibility of breaking the overall system is definitely considered a high severity bug. Any bug that falls in the middle range, not just a typo, but not enough to cause a system failure, is considered a medium severity bug.

## 7.7 Deliverables

The entire process of the code inspections will be well-documented. All the bugs will be recorded in tables in the Testing Results document, based on their respective class, organized by line number. The severity and a brief description of the bugs will also be recorded in those tables. Additionally, all of the bugs will be logged into the bug tracking system so that all of the bugs found by the team will be in one location.

# 

# 

# **Appendix A: References**

The following table summarizes the documents referenced in this document.

|  |  |  |
| --- | --- | --- |
| **Document** | **Description** | **Location** |
| Architecture | High level descriptions of core components and their interfaces | https://github.com/sampsyo/beets/wiki/Architecture |
| Testing | Wiki page describing how to run unit tests, and other information related to testing | https://github.com/sampsyo/beets/wiki/Testing |
| User Documentation | Guides for using the application | http://beets.readthedocs.org/en/latest/ |
|  |  |  |

# 

# 

# **Appendix B: Key Terms**

The following table provides definitions for terms relevant to this document.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Mutant | a small error inserted into the program. The mutant is “killed” if the program crashes or it causes at least one unit test to fail. Otherwise it is “live.” |
|  |  |
|  |  |