Beets

Testing Results

2/12/2013

TABLE OF CONTENTS

[1 Introduction](#h.frxumwlfs7cx)

[1.1 Purpose of the Testing Results Document](#h.ke9gbedj0ff)

[1.2 Audience of the Testing Results Document](#h.kixe9dfpmzgv)

[1.3 Tools](#h.nss6hlititic)

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

[2.1 Who tested?](#h.hgxsluwxowds)

[2.2 What was tested?](#h.1ljsd9k)

[2.3 How was it tested?](#h.u3xmd3kw0s5p)

[2.4 Results](#h.7o5ognilhvdj)

[2.5 Recommendations](#h.93h6ooerzgpp)

[3 Stress Testing](#h.bsrzs36gxhcd)

[3.1 Who tested?](#h.tzy6n7muo2ux)

[3.2 What was tested?](#h.stmg1axzknl7)

[3.3 How was it tested?](#h.y2dc6b6ophpj)

[3.4 Results](#h.oe9qujg3elrr)

[3.5 Recommendations](#h.4sse0ab2j6e3)

[4 Compatibility Testing](#h.ok6cpm3evvsz)

[4.1 Who tested?](#h.n5apwl63ll1q)

[4.2 What was tested?](#h.g95d8l6mzcrm)

[4.3 How was it tested?](#h.613tqtra17sh)

[4.4 Results](#h.tttqerwwx0d8)

[4.5 Recommendations](#h.6jex23yamcyb)

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

[5.1 Who tested?](#h.8ujrt8ufsumq)

[5.2 What was tested?](#h.mscqr8b6ugeb)

[5.3 How was it tested?](#h.685ekhhlgtm4)

[5.4 Results](#h.4dlnsmlbulxy)

[5.5 Recommendations](#h.ylc6dw4j6exq)

[5.6 Issues](#h.l70x6yaugep7)

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

[6.1 Who tested?](#h.mb8rczigwo74)

[6.2 What was tested?](#h.2uofsrupe196)

[6.3 How was it tested?](#h.87v8e6zcjxew)

[6.4 Results](#h.6oa672mxt5i1)

[6.5 Recommendations](#h.4zfd45ytvmdg)

[7 Code Inspections](#h.7deqnzg874yz)

[7.1 Who tested?](#h.kpm9g3f2my75)

[7.2 What was tested?](#h.8ly18bt4ng6k)

[7.3 How was it tested?](#h.3f28gposwhhs)

[7.4 Results](#h.bu32hesezn3u)

[7.5 Recommendations](#h.zforal4c4pj5)

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

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

# 

# 1 Introduction

## 1.1 Purpose of the Testing Results Document

This document states the results of the test plan executed on the the open source Beets software. The goal of these testing results is to provide an effective means of communication between the developers, the quality assurance teams, and the customer in regards to what components of the software was tested, how each component was tested, issues that were found in the software, and recommendations on how to correct said issues.

## 1.2 Audience of the Testing Results Document

Affected audience of this Testing Results document include:

* Beets Developers
* Quality Assurance Team
* 452 Software Testing Instructor
* Users of the software

## 1.3 Tools

Tools used to conduct and track the testing results:

* FreeBugBase
  + Free online defect tracking tool
  + Used to track defects found while executing the test plan
* Virtual Machines
  + Used to host various operating systems while executing the compatibility testing
  + Used to host an installation of the Beets software for testing
* Python version 2.7
  + Used for conducting unit tests
    - unittest
    - nose
  + Used for conducting coverage tests
    - coverage.py

# 

# 

# 2 User Acceptance Testing

## 2.1 Who tested?

Christopher Ketant

Eric Caron

## 2.2 What was tested?

The following Use Case diagram reveals the main interaction a user is allowed with the Beets system on a high-level listing commands but leaving out command options and choices.

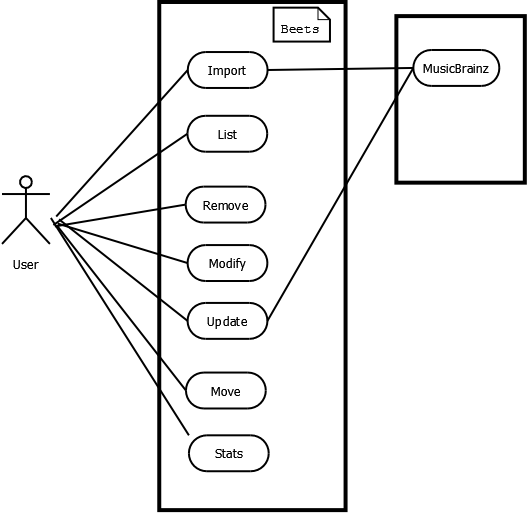


fig. 2.1 - Use Case Diagram

**Detailed user interaction with system:**

|  |  |
| --- | --- |
| ID: | 1 |
| Title: | Import |
| Description: | Importing music into the Beets library |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | Meta-data for each album, track, etc. is added, updated, or untouched |
| Main  Success Scenario: | 1. Run *beet import <option> <directory>* 2. If the command was previously ran, then the option of “*Resume (Y/n)?”* 3. Choices are then shown (see below) for performing actions on item 4. Then continue on to the next item |
| Extensions: | · **CHOICES:**  o A – Apply the suggested changes shown  o M – Show more options  o U- import the album without changing any tags  o E – User defines the search criteria instead of the automatic searching Beets uses for import  o S – Skip the Matching for the current Album or Track  o B – Abort command totally  o T – Import the directory as singleton tracks not as albums K- When there is a duplicate album in library, keep both duplicates  o R- When there is a duplicate album in library, remove old album.  · **OPTIONS:**  o a - No Auto-tagging just add the Albums and Tracks to Beets library without searching in MusicBrainz library  o w – Do not write new meta data tags for tracks that already have  o c – don’t copy imported files to music directory  o l <LOGFILE> write message to <LOGFILE> every time command is entered and output is directed to user  o q – Do not prompt user for user input, do not show user choices.  o p- automatically resume an interrupted import  o h- help, show options, arguments, suggestions  o s run in singleton mode, tagging individual tracks instead of entire albums at a time  o AC – quickly add group of files to library without tagging them  o A- noautotag – Do not automatically auto tag the music with metadata from MusicBrainz library |
| Frequency of Use: | Very Often |

|  |  |
| --- | --- |
| ID: | 2 |
| Title: | List |
| Description: | Queries the user’s music library for track or album |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from. Must be in the |
| Postconditions: | N/A |
| Main  Success Scenario: | 1. Run *ls <option> <query>* |
| Extensions: | · **OPTIONS**  o A – query only albums  o P – print out filenames of matched items  o F – lets user specify a specific format with which to print every album or track  o h- help, show options, arguments, suggestions  · **FIELDS**  o Artist:<query>- Query the music library for the Album/Track based on the artist  o Year:<query>- Query the music library for the Album/Track based on the year.  o Album:<query>- Query the music library for the Album/Track based on Album  o Title:<query>- Query the music library for the Album/Track based on Title |
| Frequency of Use: | Very Often |

|  |  |
| --- | --- |
| ID: | 3 |
| Title: | Remove |
| Description: | Remove the |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | Meta-data for each album, track, etc. is removed from metadata library and/or from disk |
| Main  Success Scenario: | 1. Run *beet remove <option> <query>*  2. If match found, confirmation message is output, *“Really remove <number of items> from the library (y/n)?”* |
| Extensions: | · **OPTIONS**  o A – query only albums  o d – deletes files from disk  o h- help, show options, arguments, suggestions |
| Frequency of Use: | Low |

|  |  |
| --- | --- |
| ID: | 4 |
| Title: | Modify |
| Description: | Supply a query matching the things you want to change and provide changes via field=value pairs. |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | Meta-data for each album, track, etc. is modified |
| Main  Success Scenario: | 1. Run *beet modify <option> <query> <FIELD=VALUE>*  2. If query yields results then prompt is initiated asking to change |
| Extensions: | · **OPTIONS**  o A – queries album  o M – Disable moving items  o w- override settings in configuration file that are dedicated to importing  o W- don’t write tags  o y- skip prompt  o h- help, show options, arguments, suggestions |
| Frequency of Use: | Medium |

|  |  |
| --- | --- |
| ID: | 5 |
| Title: | Move |
| Description: | Move or copy items in your library |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | Item is moved or copied over to new directory |
| Main  Success Scenario: | 1. Run *beet move <option> <query>* |
| Extensions: | · **OPTIONS**  o d – destination directory  o c – copies files instead of moving  o a – matches albums  o h- help, show options, arguments, suggestions |
| Frequency of Use: | Medium |

|  |  |
| --- | --- |
| ID: | 6 |
| Title: | Update |
| Description: | Update the library to reflect metadata changes and file deletions |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | Metadata for tracks and albums updated |
| Main  Success Scenario: | 1. Run *beet update <option> <query>* |
| Extensions: | · **OPTIONS**  o a- matches album  o m- disabling files being named according to their new metadata  o p – show all proposed changes but won’t actually change anything on disk  o h- help, show options, arguments, suggestions |
| Frequency of Use: | Often |

|  |  |
| --- | --- |
| ID: | 7 |
| Title: | Stats |
| Description: | Show statistics on library |
| Primary Actor: | User |
| Preconditions: | Music files must be present in the directory from which you are importing from |
| Postconditions: | n/a |
| Main  Success Scenario: | 1. Run *beet stats <option> <query>* |
| Extensions: | · **OPTIONS**  o e – makes the calculation of total file size accurate  o h- help, show options, arguments, suggestions |
| Frequency of Use: | Medium |

|  |  |
| --- | --- |
| ID: | 8 |
| Title: | Help |
| Description: | Shows list of commands and option |
| Primary Actor: | User |
| Preconditions: | n/a |
| Postconditions: | n/a |
| Main  Success Scenario: | 1. Run *beet help* or run *beet -h <command>* |
| Extensions: | o h- help, show options, arguments, suggestions |
| Frequency of Use: | Very Often |

**Acceptance Tests:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID/Command** | **Description** | **Expected Results** | **Actual**  **Results** |
| beet help | Pre-condition: Beets installed. Desc: Run beet help | Help output displayed |  |
| beet import | Pre-condition:Beets installed. Desc: Run beet import | Nothing |  |
| beet import <directory> | Pre-condition: directory contains music files Desc: Run beet import | import music files while adding choices for action at each item |  |
| beet import -A <directory> | Pre-condition: directory contains music files Desc: Run beet import -A | import music files without autotagging them |  |
| beet import -s <directory> | Pre-condition: directory contains music files Desc: Run beet import -s | import music files by processing tracks individually |  |
| beet import -i <directory> | Pre-condition: directory contains music files Desc: Run beet import -i | import music files while skipping music files already imported |  |
| beet import -q <directory> | Pre-condition: directory contains music files Desc: Run beet import -q | Imports files without prompting user for actions |  |
| beet ls | Pre-condition: directory contains music files Desc: Run beet ls | List all files within library |  |
| beet ls <query> | Pre-condition: directory contains music files Desc: Run beet ls <query> | List all files associated with word(s) <query> |  |
| beet ls -a <query> | Pre-condition: directory contains music files Desc: Run beet ls -a <query> | List album associated with <query> |  |
| beet ls <field>:<query> | Pre-condition: directory contains music files Desc: Run beet ls <field>:<query> | List query associated with <field> (i.e. artist) : <query> |  |
| beet remove | Pre-condition: directory contains music files Desc: Run beet remove | Remove all items from library |  |
| beet remove <query> | Pre-condition: directory contains music files Desc: Run beet remove <query> | Remove only items associated with query |  |
| beet remove -d <query> | Pre-condition: directory contains music files Desc: Run beet remove -d <query> | Removes files from disk |  |
| beet modify | Pre-condition: directory contains music files Desc: Run beet modify | nothing | error message, "error message: no modification specified" |
| beet modify <query> | Pre-condition: directory contains music files Desc: Run beet modify <query> | nothing | error message, "error message: no modification specified" |
| beet modify <query> <field>=<value> | Pre-condition: directory contains music files Desc: Run beet <query> <field>=<value> | query is modified at its field |  |
| beet modify -W<query> <field>=<value> | Pre-condition: directory contains music files Desc: Run beet -W <query> <field>=<value> | Modification not to be written written to Beets Library | Written to Beets library but not written to disk (ambiguous requirement) |
| beet move | Pre-condition: directory contains music files Desc: Run beet move | nothing |  |
| beet move <query> | Pre-condition: directory contains music files Desc: Run beet move <query> | moves query | Does Nothing |
| beet move -d <path to dir> <query> | Pre-condition: directory contains music files Desc: Run beet move -d <query> | moves query to <path to dir> |  |
| beet move -c <query> | Pre-condition: directory contains music files Desc: Run beet move -c <query> | Copy's file to current location |  |
| beet update | Pre-condition: directory contains music files Desc: Run beet update | Updates all files |  |
| beet update <query> | Pre-condition: directory contains music files Desc: Run beet updaet <query> | Updates <query> |  |
| beet update -p <query> | Pre-condition: directory contains music files Desc: Run beet ls <query> | Shows prompt to user for possible updates |  |
| beet stats | Pre-condition: directory contains music files Desc: Run beet stats | Dispaly statistics of library |  |
| beet stats -e | Pre-condition: directory contains music files Desc: Run beet stats -e | Display total statistics of library |  |
| beet stats <query> | Pre-condition: directory contains music files Desc: Run beet stats <query> | Display statistics of <query> |  |

## 2.3 How was it tested?

There are 8 main commands for the Beets command line system with each command owning different number of options. Also considering the different states that the user’s music library could be in whether filled, empty, a single album, single track, non-music related files, etc. the possible number of test cases raises exponentially.

The rationale for Acceptance Testing is equivalence partitioning in order to achieve a reasonable most effective Acceptance Test while focusing on boundary cases. After this equivalence partition was identified for each command then the test were run.

Prior to running the Acceptance Test Use Case diagrams and Use Case cards were created in order to layout all possible interactions available to the user.

## 2.4 Results

Since a formal requirements document was not available the test cases were compiled manually by searching tutorials, asking the main developer, and extracting from the documentation available. The expected results, usually provided by requirements document, was created by myself based on intuition and assumption. This made it difficult to exactly determine the expected results as compared to the actual result itself.

## 2.5 Recommendations

Differentiating between directories that are potential candidates for imports and directories that are not. When memory usage approaching max warn the user, especially since nearly every command uses up memory. Disambiguate documentation. Add more formal user case examples and functional requirements.

# 

# 

# 3 Stress Testing

## 3.1 Who tested?

Shun Mok Bhark

## 3.2 What was tested?

The robustness and availability of the software beyond its normal limits of operation.

## 3.3 How was it tested?

Tested the functionality of the software by taking away resources it would normally run on. The testing was conducted on a Debian 6 virtual machine with 32 mb of RAM. The virtual machine was running a heavily optimized Debian 6 operating system to ensure that no other processes would have an impact on the testing and to allow the operating system to run on minimal amounts of RAM.

A series of installations and imports were ran on the resource bare virtual machine.

## 3.4 Results

After running the series of installations and imports, the functionality and user experience was compared with the same operations ran on a resource generous machine. The results were similar in that Beets performed at the same capability it would under normal operation when run on a machine with minimal resources.

## 3.5 Recommendation

Keep the code as is so that it can be run under minimal resource scenarios.

# 

# 

# 4 Compatibility Testing

## 4.1 Who tested?

Christopher Norman

## 4.2 What was tested?

The compatibility of the Beets software against various operating systems. Those systems include Mac OS X, Windows, and Linux.

## 4.3 How was it tested?

Python was installed on the various operating systems and then the Beets software was downloaded and installed. If success appeared after the software was installed It made sure the software worked by loading in some music. If that that then worked it was deemed a successful install.

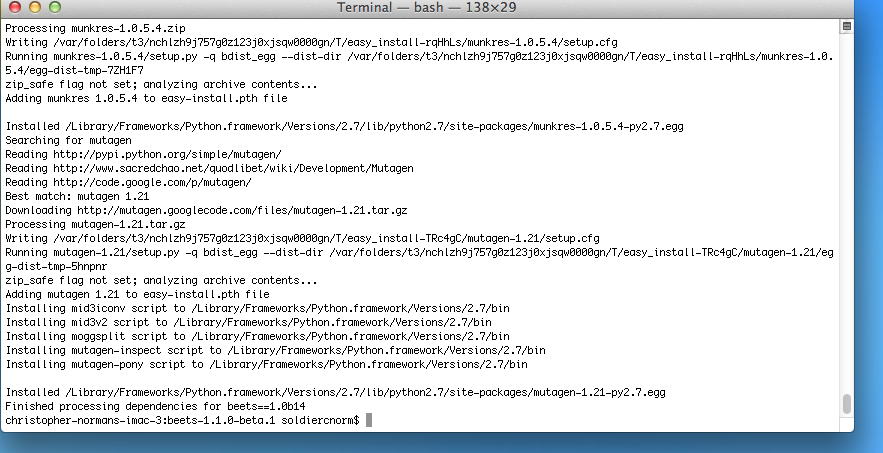
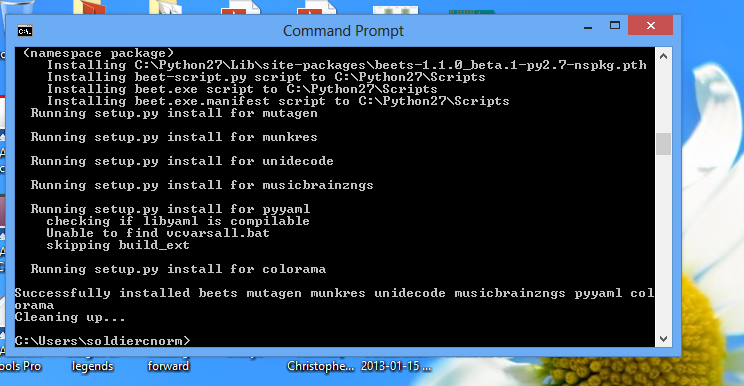
## 4.4 Results

|  |  |  |
| --- | --- | --- |
| **Type of OS** | **Results** | **Pass/Fail** |
| Mac OS X 10.5 | Smooth installation | Pass |
| Mac OS X 10.6 | Had to download extra python packages, but installed correctly | Pass |
| Mac OS X 10.7 | Smooth installation once python was configured correctly | Pass |
| Mac OS X 10.8 | Smooth installation | Pass |
| Windows XP | Had to do configuration to the os environment, but it installed correctly | Pass |
| Windows Vista | Had to do configuration to the os environment, but it installed correctly | Pass |
| Windows 7 | Had to do configuration to the os environment, but it installed correctly | Pass |
| Windows 8 | Had to do configuration to the os environment, but it installed correctly | Pass |
| Ubuntu | Smooth installation | Pass |
| RedHat | Smooth installation | Pass |

## 

## 4.5 Recommendations

The main recommendation is to possibly find a way to make the installation on Windows easier. I found installing onto the various windows operating systems was tougher because you had to configure various parts of the operating system environment.



# 

# 

# 5 Mutation & Coverage Testing

## 5.1 Who tested?

## David Wilson

## 5.2 What was tested?

The existing test suite was tested for coverage.

## 5.3 How was it tested?

Coverage testing was done by running coverage.py against the test suite. Data was collected using the command:  
 *coverage run --source beets test/testall.py*

Then, the data was organized into a useful format using the command:  
 *coverage html*

This data was sorted by coverage percent and by number of missed statements to identify particularly uncovered areas.

## 5.4 Results

Initial Coverage Testing Results: <http://vm452-3.se.rit.edu/htmlcov-original/>

Large Deficiencies:

|  |  |  |
| --- | --- | --- |
| **File** | **Coverage %** | **Statements Missed** |
| util/bluelet | 26% | 223 |
| beetsplug/bpd/\_\_init\_\_ | 27% | 447 |
| util/artresizer | 39% | 49 |
| ui/commands | 51% | 327 |
| importer | 62% | 192 |
| ui/\_\_init\_\_ | 70% | 133 |

Final Coverage Testing Results: <http://vm452-3.se.rit.edu/html_latest_clone_updated/>

## 5.5 Recommendations

ui/commands, ui/\_\_init\_\_, and importer are obvious areas for improvement. These are also critical components of the user experience. Developers seem to be doing well at creating tests for new code, but attention should also be paid to technical debt and to how new code impacts existing code and tests.

## 5.6 Issues

PyMuTester, the only mutation testing utility compatible with this project, stopped abruptly and quietly during operation and produced no meaningful output. The potential benefits of mutation testing were deemed not significant enough to merit spending the time to debug PyMuTester itself.

# 6 Unit Testing

## 6.1 Who tested?

Shun Mok Bhark

## 6.2 What was tested?

All unit tests in the test folder were tested. The unit test for testing UI commands was more heavily tested as it lacked coverage and was an important area of functionality.

## 6.3 How was it tested?

The command: python -3 testall.py, was used to execute all unit tests in the test folder.

The command then produce an output of how many test errors and failures existed in the program.

## 6.4 Results

At the latest pull of the program code, the unit tests contained a total of six failures. The failures were not critical as it looked like some of the assertions required some updates. The unit tests that contained the faulty assertions were not updated due as it was newly introduced failures in comparison with an older build on our test VM. As the developers seemed to be active in committing and updating the unit tests we decided to leave the failed unit tests for the developers to resolve.

We were able to add some additional unit tests for the UI commands. We specifically tested the importing of music into a library as we deemed this was an important feature of the program but was not covered by existing unit tests. After the addition of our unit tests we were able to increase the coverage of the UI commands by 3% while the total coverage of all the unit tests was increased by 1%. We believe that this was a great achievement in increasing the total unit test coverage by 1%.

## 6.5 Recommendations

Although the files of unit tests were well named there was a lack of comprehensive comments in the unit tests. I found I had to step through the tests multiple times across multiple unit tests to identify the logic of the tests. If there were more comments to the unit tests code it would be easier for future developers to add contribute to the testing of the beets software.

# 7 Code Inspections

## 7.1 Who tested?

Moderator: Jenny Zhen

Scribe: Eric Caron

Reader: Chris Ketant

Tester: David Wilson

Author: Adrian Sampson (was not present)

## 7.2 What was tested?

The team conducted code reviews for 4 classes in the beets/autotag directory. Those classes were \_\_init\_\_.py, hooks.py, match.py, and mb.py.

## 7.3 How was it tested?

The moderator planned a time and location for a face-to-face meeting in the SE team rooms. The team, then, decided which classes needed to be review. Since the autotag feature was one of the most important features, the team decided to review the classes for that feature. Prior to the meeting, the moderator required team members attending the code review to annotate a printed copy of the code for anything to be discussed. During the meeting, the moderator and reader led the team through each class by section and the team discussed any possible issues. Anything considered a bug was written down by the scribe, with the line number and severity of the bug. The solution to the bugs were not discussed because the goal of the code review was only to find bugs.

## 7.4 Results

\_\_init\_\_.py

|  |  |  |
| --- | --- | --- |
| Lines | Bug | Severity |
| 25-29 | The import style is different from the other imports in files in the beets/autotag directory. | Low |
| 47 | Collapse\_pat is named poorly; may be easily confused with collapse\_paths. | Medium |
| 160 | “apply\_metadata” is ambiguous. It should be “apply\_album\_metadata”, as it is the album counterpart to “apply\_item\_metadata”. | Medium |
| 166-205 | Some code from apply\_item\_metadata is duplicated here. Utilize this function for each album. | Medium |

hooks.py

|  |  |  |
| --- | --- | --- |
| Lines | Bug | Severity |
| 27-106 | AlbumInfo and TrackInfo should extend a parent class to consolidate common attributes and the python-musicbrainz-ngs bug workaround. | Medium |
| 64-86 | There are too many arguments in the constructor. Using a dictionary may improve readability and maintainability. | Low |
| 129-139 | There are too many arguments in the constructor. Using a dictionary may improve readability and maintainability. | Low |

match.py

|  |  |  |
| --- | --- | --- |
| Lines | Bug | Severity |
| 39 | Typo in comments (“umatched”). | Low |
| 77 | “RECOMMEND\_LOW” isn’t consistent with “RECOMMEND\_STRONG” and may cause confusion as it is associated with high distances. “RECOMMEND\_WEAK” would be better. | Medium |
| 326-328 | Missing “RECOMMEND\_LOW” in comments. | Low |
| 389-392 | Missing “RECOMMEND\_LOW” in comments. | Low |
| 335, 343 | Possible logic error; “<” compared to “<=”. | Medium |
| 395 | AutotagError was mentioned in comments, but was not raised anywhere in the method. | High |

mb.py

|  |  |  |
| --- | --- | --- |
| Lines | Bug | Severity |
| 22 | The entire beets directory is imported. | High |
| 173 | Typo in comments (“underling”). | Low |

## 

## 7.5 Recommendations

There are occasional typos that a spell checker might not catch, but those are in the comments. The author should proofread the comments after they are written. The general naming conventions are good, but the two instances mentioned above can be fixed. There is also a lot of, what seems to be, duplication in the code. Certain methods can be combined. Occasionally, magic numbers are used when indexing (not reported in the list of bugs). Constants may be used to reference specific indices.

# **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.” |