

*Software Engineering*

Smart Robo-Advisor

TESTING

**GROUP REPORT**

1801212963 Ye Juelin (叶珏琳)

1801212938 Wang Rusi (王如思)

1801212920 Shi Yunxia (史云霞)

1801212932 Wang Bingquan (王柄荃)

1801212832 Shen Zimin (谌子民)

# CATALOG

[1. Introduction 3](#_Toc4962401)

[2. Test strategy 4](#_Toc4962402)

[2.1 Overall strategy 4](#_Toc4962403)

[2.2 Test range 5](#_Toc4962404)

[2.3 Test handover standards 6](#_Toc4962405)

[2.4 Test pass criteria 6](#_Toc4962406)

[2.5 Test type 6](#_Toc4962407)

[2.6 Risk Analysis 8](#_Toc4962408)

[3. Test plan 8](#_Toc4962409)

[4. Test Case Specification 10](#_Toc4962410)

[4.1 Investor risk assessment algorithm 10](#_Toc4962411)

[4.2 Multi-factor Strategy Algorithm 11](#_Toc4962412)

[4.3 Style Rotation Strategy Algorithm 12](#_Toc4962413)

[4.4 Database Downloading Function 14](#_Toc4962414)

[4.5 CTA Strategy Algorithm 17](#_Toc4962415)

# Introduction

Software testing is a process to promote the validity, integrity, security and quality of software. The purpose is to examine or compare an actual output with an expected output.

Our testing targets are as follows:

1) Discover development risks that can be avoided through testing.

2) Implement tests to reduce the risks found.

3) Determine when the test can be completed.

4) Consider testing as a standard project in the process of developing a project.

Our test objects include programs, data, and file.

In this document, we utilize **Unit Testing IN PYTHON 3.0 by importing unittest Library** to test the correctness of the smallest unit of software design - program module. Its purpose is to discover all kinds of errors that may exist in each module.

The advantages of unit testing include:

1) Easy to reconstruct later. Unit testing can provide protection for code refactoring. As long as the unit tests run well after the refactoring, it means to a large extent that the refactoring does not introduce a new BUG. Of course, it is based on the complete and effective coverage of unit testing.

2) Optimum design. Writing unit tests will enable users to observe and think from the caller's point of view, especially using the development method of TDD-driven development, which will enable users to design programs that are easy to call and test, and to decouple the software.

3) Document recording. Unit testing is an invaluable document that shows how functions or classes can be used. This document is compiled, runnable, up-to-date, always in sync with the code.

4) Regressive. Automated unit testing avoids code regression. After writing, you can quickly run the test anywhere and anytime, instead of deploying the code to the device, and then manually covering various execution paths. This behavior is inefficient and wastes time.

# Test strategy

## 2.1 Overall strategy

The characteristics of this project:

1) Part of team members are the first time to develop a system

2) The project system is huge, the content is extensive, and the functions are complex.

3) The distance from the initial inspection time is tight.

Based on the above characteristics, the test strategy for developing this project is as follows:

1) Try to find as many defects as possible (especially serious defects) in a limited time.

2) The test plan and part of the use case design are synchronized.

3) The testing process are controlled. Tests are performed according to a predefined test execution sequence, and a test record sheet is filled in to ensure that the test process is controlled.

4) Determine the focus. The test focuses on the function implementation of each subsystem.

Standard:

In this test, the preparation of test documents, the preparation of test cases, the specific execution test, and the allocation and estimation of various resources in the test are all based on the requirements documents and design documents of each subsystem provided by the project manager. Execution is based on the system logic design architecture.

## 2.2 Test range

The basis for formulating the scope of the project test is:

1) The functions contained in each subsystem.

2) Test scope specifically determined by the project leader.

The subsystem to be tested:

Tablel 2‑1 Test range

|  |  |
| --- | --- |
| Test content | Test range |
| function test | Multi-factor test |
| CTA test |
| Style rotation test |
| Risk assessment test |
| Database downloading test |

## 2.3 Test handover standards

**2.3.1 Unit test handover standard**

This test is mainly carried out by the project team, and each project team can make an agreement according to the project needs.

**2.3.2 Integration test handover standard**

This test is performed by the project team and the test team.

The agreed test content is completed and passed the unit test, which is confirmed by the relevant personnel.

## 2.4 Test pass criteria

1) The planned test cases have all been implemented.

2) All defects identified have been agreed upon and no new defects have been identified.

## 2.5 Test type

**2.5.1Functional Testing**

Table 2-2 Description of functional test type

|  |  |
| --- | --- |
| Test Objectives | Verify that the functionality provided by the software is achievable. |
| Test methods and techniques | Verify that the results are consistent with the design expectations when entering the correct data;  Verify that the software can alarm and operate normally when entering erroneous data. |
| Standard | All functions are tested and reach the goal. |

**2.5.2 Performance Test**

Table 2-3 Performance Test Type Description

|  |  |
| --- | --- |
| Test Objectives | Test the software's response time, concurrency, throughput, processing accuracy and other indicators to confirm whether the software meets customer needs. |
| Test methods and techniques | Test each function and record it using the black box method. |
| Standard | All indicators meet the standards. |

**2.5.3 Capacity test**

Table 2-4 capacity test type description

|  |  |
| --- | --- |
| Test Objectives | Increase the amount of software input data to confirm that the software is functioning properly when processing large amounts of data. |
| Test methods and techniques | Increase the average amount of data processing expected at design time by an order of magnitude. |
| Standard | The software can complete the function normally when the standard large data input is completed. |

**2.5.4 Safety test**

Table 2-5 Security Test Type Description

|  |  |
| --- | --- |
| Test Objectives | Ensure that software users are operating within their authority. |
| Test methods and techniques | Black box testing is used to ensure security by logging in to different privileged user modes for software operations. |
| Standard | Each privileged user can only operate within the scope of the privilege. |

## 2.6 Risk Analysis

1) The risk of the tester's familiarity with the system:

After the short-term system training, the testers participating in the project may still not fully grasp the business details of the system, which will cause some test escape phenomenon in the later test design and test execution work. (ie some aspects to be tested are not covered)

2) Risks of testing tools:

At present, testers have insufficient experience in using some tools for performance testing and it takes a certain amount of time to study.

# Test plan

Time requirements and staffing arrangements：

|  |  |  |
| --- | --- | --- |
| Task details | Time requirements (days) | personnel |
| Pre-test task preparation | 2 | 2 |
| Test plan writing | 2 | 2 |
| Test case writing | 2 | 2 |
| Function testing | 3 | 2 |
| Integration Testing | 3 | 2 |
| System test | 3 | 2 |

We divided the entire testing process into the following milestones:

Table 4-1 Test Process Information Table

|  |  |
| --- | --- |
| Milestones | **Completion criteria** |
| System Training: | 1. Complete the training for all systems in this project that need to be tested.  2. The tester has used all the systems/modules under test to understand the specific functions of the system under test. |
| Test Design: | 1. Test cases have covered all test requirements.  2. The test case design has been completed. |
| Test execution: | 1. All test cases are executed.  2. The defects found have a defect record.  3. The test process has a test record. |
| Analysis of results: | 1. Complete the test analysis report. |

# Test Case Specification

## 4.1 Investor risk assessment algorithm

|  |  |
| --- | --- |
| Test Case ID: | 01 |
| Title: | Investor risk assessment algorithm |
| Purpose: | Assess the risk-taking level of investors from aspects of age, income flow, investment experience, asset liquidity, financial knowledge, investment aim, risk tolerance level etc. |
| Initial Conditions: | Users input 9 risk feature choices from the investor risk assessment questionnaire from the interface. |
| Test Data: | 1. self.assertEqual(0.5, risk\_evaluation('A','A','C','D','A','A','A','B','D'))  2. self.assertNotEqual(0.5, risk\_evaluation('A','A','C','D','A','A','A','A','D')) |
| Test Actions: | According to the risk characteristics of user input, the system calculates and returns its corresponding risk value (between 0 to 1):  Return True if it is the same as the pre-set Equal value  Return True if it is not the same as the pre-set NotEqual value |
| Expected Results: |  |

## 4.2 Multi-factor Strategy Algorithm

|  |  |
| --- | --- |
| Test Case ID: | 02 |
| Title: | Multi-factor Strategy Algorithm |
| Purpose: | Process the factors, combine the factors, select stocks and calculate their weights, and get the net asset value of the hedged portfolio |
| Initial Conditions: | Input the benchmark for hedging, buying cycle, cost coefficient. |
| Test Data: | The fundamental and technical data of A share stocks. |
| Test Actions: | Because the output of the algorithm is complicated, we test several methods in it.   1. self.assertEqual(d.bench, "000905.SH"): test the initial part of the class. 2. self.assertEqual(d.nstock, 3598): test whether the number of stock included in the data is consistent with our expectation. 3. self.assertTrue(all(d.get\_ic(d.market\_value, 5).dropna().abs() <= 1)): test whether the absolute value of ic is lower than 1. 4. self.assertTrue(all(d.normalize(d.market\_value).sum(axis=1).abs() < 10\*\*(-10))): test whether the sum of the normalized data is equal to 0. 5. self.assertTrue(all(d.get\_combine\_factor(factor) == factor)): test whether the factor after combined is equal to the initial one if we just combine one factor. 6. self.assertTrue(all(d.stock\_selection\_and\_position(factor).sum(axis=1) == 1)): test whether the sum of the position is equal to 1. |
| Expected Results: |  |

## 4.3 Style Rotation Strategy Algorithm

|  |  |
| --- | --- |
| Test Case ID: | 03 |
| Title: | Style Rotation Strategy Algorithm |
| Purpose: | Seek for the best investment style by comparing different factors of index. Based on this, investing on the most promising index to get alpha profit. |
| Initial Conditions: | Users input the start date and the end date of back test in the interface. |
| Test Data: | Close price data of different index which are obtained from our local database. |
| Test Actions: | Considering this strategy is performed step by step, we test the whole algorithm by testing different function of the algorithm.   1. self.assertEqual(len(data.columns), 3):test whether the data contain close price data of three index. 2. self.assertTure(all([(i in [0,1,-1]) for i in get\_signal(data).signal.values]):test whether every day the strategy can give a correct trading signal. 3. self.assertTure(all(abs(data.daily\_profit.values-1)) < 0.5):test whether the profit of every day is calculated correctly. 4. self.assertTure(all(data.net\_value.values-1 >= 2)):test whether the net value of the strategy is calculated correctly. |
| Expected Results: |  |

## 4.4 Database Downloading Function

|  |  |
| --- | --- |
| Test Case ID: | 04 |
| Title: | Database Downloading Function |
| Feature/Subfeature: |  |
| Purpose: | To test whether this functions could download quota data of stocks and stock indexes from Wind or Tushare and whether it could update the data to the latest version. |
| Initial Conditions: | 1.Developers input the list of stocks or stock indexes, starting date and ending date of the quota they want to download from Wind or Tushare.  2.Developers run the update function to get the latest data. |
| Test Data: | Quota data of stocks and stock indexes in the database. |
| Test Actions: | There are several functions for downloading different types of data or from different sources. We test those data independently.   1. self.assertEqual(getindexquota(['000001.SH','000016.SH','000012.SH'], '2010010', '20190326').columns, ['000001.SH','000016.SH','000012.SH']):test whether the all stock indexes are downloaded as we want, and whether the data is saved in the type of DataFrame (index: trade date; columns: list of indexes). 2. self.assertEqual(getindexquota(['000001.SH','000016.SH','000012.SH'], '20100101'， '20190326').index,trade\_date('20100101','20190326')):test whether the data of stock indexes in this time period is downloaded as we want 3. self.assertEqual(getistockquota(['000001.SZ','000004.SZ','000060.SZ'], 20100101,20190326).columns, ['000001.SZ','000004.SZ','000060.SZ']):test whether the all stocks are downloaded as we want, and whether the data is saved in the type of DataFrame(index: trade date; columns: list of stocks ). 4. self.assertEqual(getstockquota(['000001.SZ','000004.SZ','000060.SZ'], '20100101'， '20190326').index,trade\_date('20100101','20190326')):test whether the data of stocks in this time period is downloaded as we want 5. self.assertEqual(updateindexquota().index[-1],'20190329'):test whether the data of indexes are updated to the latest trade\_date. 6. self.assertEqual(updatestockquota().index[-1],'20190329';):test whether the data of indexes are updated to the latest trade\_date |
| Expected Results: | Test method getindexquota ( stock\_list, start\_date, end\_date) ... ok  Test method getindexquota ( stock\_list, start\_date, end\_date) ... ok  Test method getstockquota ( stock\_list, start\_date, end\_date) ... ok  Test method getstockquota ( stock\_list, start\_date, end\_date) ... ok  Test method updateindexquota ( stock\_list, start\_date, end\_date) ... ok  Test method updatestockquota ( stock\_list, start\_date, end\_date) ... ok  ----------------------------------------------------------------------  Ran 6 tests in 0.001s  OK |

## 4.5 CTA Strategy Algorithm

|  |  |
| --- | --- |
| Test Case ID: | 05 |
| Title: | CTA strategies |
| Purpose: | 1.To test whether the strategy could read the quota data of futures from database  2. To test whether the strategy could find the trading signals  3. To test whether the strategy could output the back-test result such as net value of the asset. |
| Initial Conditions: | 1. Input the time period for trading  2. Input the type of futures for trading |
| Test Data: | Quota data of futures in database. |
| Test Actions: | Because there are several functions run step by step in this algorithm, we test them step by step.  1. self.assertEqual(readquota(' Y0607.DCE', '20180101','20190320').columns,'' Y0607.DCE''): test whether the strategy could get the needed data from database and whether it is saved in DataFrame.  2. self.assertEqual(readquota(' Y0607.DCE', '20180101','20190320').index, trade\_date('20180101','20190320')): test whether the strategy could get the data of all trading days during this period.  3. self.assertEqual(len(getsignal('RSI')) > 0)): test whether there are several signals during this time period.  4.self.assertEuqal(return('RSI').index, trade\_date('20180101','20190320')): test whether there is a return in all days during this time period |
| Expected Results: | Test method readquota(future, start\_date, end\_date) ... ok  Test method readquota(future, start\_date, end\_date) ... ok  Test method getsignal(strategy) ... ok  Test method return(strategy) ... ok  ----------------------------------------------------------------------  Ran 4 tests in 0.001s  OK |