***Algorithmic Trading System***

***Software Test Plan***

Okanagan College

Algorithmic Trading System

2024-04-09

Version 1.3

Authors: Jacob Rawlings,Jake Fischer**,** Alan Abdollahzadeh, Devon Volberg, William Blake, Benjamin Carrier, Vanessa Dubouzet, Dakota Flath, Parker Green, Yuan Hu, Sachin Kamboj, Jacob Labelle, Isaac Lengacher-Bergeron , Dominic Presch, Gurneer Sandhu, Dylan Soares, Jaeden Soukoroff

# Contents

[**Contents 2**](#_heading=h.gjdgxs)

[**Revision Sheet 2**](#_heading=h.30j0zll)

[**1. Introduction 1**](#_heading=)

[**2. Scope 1**](#_heading=)

[2.1 Identification 1](#_heading=h.tyjcwt)

[2.2 Document Overview 1](#_heading=)

[2.3 Acronyms and Definitions 2](#_heading=)

[2.3.1 Acronyms 2](#_heading=)

[2.3.2 Definitions 2](#_heading=)

[**3. Referenced Documents 4**](#_heading=h.41mghml)

[**4. Test Identification 5**](#_heading=)

[4.1 General Information 5](#_heading=h.vx1227)

[4.1.1 Test Levels 5](#_heading=h.3fwokq0)

[4.1.2 Test Classes 8](#_heading=)

[4.1.3 General Test Conditions 9](#_heading=h.1yyy98l)

[4.2 Planned Testing 9](#_heading=)

[4.2.1 Module Test 9](#_heading=)

[**5. Test Schedules 10**](#_heading=)

[**6. Risk Management 11**](#_heading=)

6.1 [Third-Party Dependencies 11](#_heading=h.2pta16n)

6.2 [System Requirements 12](#_heading=h.38czs75)

6.3 [Availability of Resources 12](#_heading=h.11si5id)

6.4 [Alignment of Development and Testing Environments 12](#_heading=h.3dhjn8m)

[**7. Requirements Traceability 13**](#_heading=)

# Revision Sheet

| Revision Number | Date | Brief summary of changes |
| --- | --- | --- |
| Version 1.0 | 2023-10-22 | Initial document |
| Version 1.1 | 2023-12-07 | Document update |
| Version 1.2 | 2024-02-14 | Updating with ui and dw changes |
| Version 1.3 | 2024-04-09 | Updates to reflect changes to Python script testing |

## Introduction

This document outlines a comprehensive testing method for the Algorithmic Trading software project. The system is designed to collect and process financial market data from various sources, process that data, and make it available for machine learning models. This test plan provides a framework to ensure that the system's functionality is in line with project requirements.

## Scope

### *Identification*

The software test plan outlines the testing strategy for version 1.0 of the Algorithmic Trading System (ATS). Testing guarantees that the clients requirements are being met, and verifies that the software is performing correctly.

The goal of the ATS is to collect, clean, and store high-quality financial data. This software will allow machine learning models to be trained on large datasets to increase forecast accuracy. Specifications regarding the softwares modules and functionalities are detailed in the SMF Software Requirements Specification (SRS) [R1], and are directly traceable in the Requirements Traceability Matrix (RTM). These documents connect requirements to their respective project elements.

### *Document Overview*

This document outlines the Software Test Plan (STP) for the Algorithmic Trading System. ATS documentation will include references to:

* System Requirement Specification (SRS)
* Requirements Traceability Matrix (RTM)
* Software Design Document (SDD)
* Software Configuration Management Plan (SCM)
* The ATS Technical Manual

These documents offer additional context for testing the software effectively

**Software components to be tested**

* Data Collection scripts
* Database insertion scripts
* Utility scripts
* User Interface functionality
  + Includes system configuration management

**Software features to be tested**

* Data quality validation
* Data accessibility
* System availability

### *Acronyms and Definitions*

#### *Acronyms*

| RTM | Requirements Traceability Matrix |
| --- | --- |
| SCM | Software Configuration Management |
| SDD | Software Design Document |
| SOW | Statement of Work |
| SPP | Software Project Planning |
| QA | Quality Assurance |
| SRS | Software Requirements Specification |
| STP | Software Test Plan |
| STR | Software Test Report |
| STD | Software Test Description |
| CER | Change Enhancement Request |
| TPR | Test Problem Report |
| ITR  ATS  CI | Internal Test Report  Algorithmic Trading System  Continuous Integration |
|  |  |

#### *Definitions*

| **baseline** | “(1) A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures. (2) A document or a set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. *Note:* Baselines, plus approved changes from those baselines, constitute the current configuration identification. (3) Any agreement or result designated and fixed at a given time, from which changes require justification and approval.”[IEE91] |
| --- | --- |
| **configuration management** | “A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing implementation status, and verify compliance with specified requirements.” [IEEE90] |
|  |  |
| **implementation** | “(1) The process of translating a design into hardware components, software components, or both. (2) The result of the process in (1).”[IEEE91] |
|  |  |
| **integration test** | “Testing in which software components, hardware components, or both are combined to evaluate the interaction between them.”[IEEE91] |
| **module test** | --(1)A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (2) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item. |
| **plan** | “A detailed scheme, program, or method worked out beforehand for the accomplishment of an objective.” [Heritage85]  —Defined set of procedures and the required resources to implement a policy. |
| **policy** | “A course of action, guiding principle, or procedure considered to be expedient.” [Heritage85] |
|  | —Corporate strategy, defines high-level goals. |
| **process** | “A sequence of steps performed for a given purpose.” [IEEE90] |
|  | —Activities and interfaces used to implement the plan. |
| **project** | —unit of work to meet a specific customer requirement. Includes all tasks, activities, and functions necessary to meet the requirements. |
| **project deliverables** | “The work product(s) to be delivered to the customer. The quantities, delivery dates, and delivery locations are specified in the project agreement.” [IEEE87] |
| **quality assurance** | “(1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.” [IEEE90] |
|  | “(2) A set of activities designed to evaluate the process by which products are developed or manufactured. ” [IEEE90] |
| **review** | —A process or meeting during which a work product, or set of work products, is presented to program personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, test readiness review. |
| **software** | “Computer programs, procedures, and associated documentation and data pertaining to the operation of a computer system.” [IEEE90] |
| **software engineering** | “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.” [IEEE90] |
| **software project management** | “The process of planning, organizing, staffing, monitoring, controlling, and leading a software project.” [IEEE87] |
| **Software Test Plan** | -- A document that specifies the test inputs, execution conditions, and predicted results for an item to be tested. |
| **specification** | “A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.” [IEEE90] |
|  |  |
| **system requirements** | —A condition or capability that must be met or possessed by a system or subsystem component to satisfy a condition or capability needed by a user to solve a problem. |
| **task** | “The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. The specification of work to be accomplished in completing a task is documented in a work package. Related tasks are usually grouped to form activities.” [IEEE87] |

## Referenced Documents

* [R1] IEEE98 IEEE Std 829, Standards for Software Documentation
* [R2] System Requirement Specification (SRS)
* [R3] Requirements Traceability Matrix (RTM)
* [R4] Software Design Document (SDD)
* [R5] Software Configuration Management Plan (SCM)
* [R6] Change Request Policy
* [R7] The AT Technical Manual

## Test Identification

### *General Information*

#### *Test Levels*

The successful development and delivery of high-quality software depends on a comprehensive testing strategy. It’s essential to verify that the software meets specified requirements and functions correctly. Three primary types of tests should be performed throughout the entire development process: unit testing, integration testing, and system testing. Each of these tests plays a distinct role in ensuring the reliability and functionality of the software.

These test levels serve as a structured approach to assess the software at different stages of development, starting from the smallest code units and extending to the complete system. The goal of this structured approach is to identify and rectify defects and inconsistencies early in the development cycle, minimizing the potential for critical issues to reach production.

##### **Unit Testing**

##### **Scope**

Unit testing is the initial layer of the testing process, focusing on individual components and modules that make up the software. At this level, the primary goal is to assess the correctness of isolated units of code.

##### **Objective**

The objective of unit testing is to meticulously examine and verify the correctness of individual functions, procedures, and modules within the software. It is aimed at ensuring that each unit of code, such as functions or modules, behaves as expected and fulfills its designated purpose without introducing errors or unexpected behaviors.

***Testing Approach***

The development team will write and execute tests using unit testing tools to verify the behavior of each unit of code (functions, procedures, modules); for example, the return of a function within a larger module should be compared against the expected results to assert correctness.

Test cases are designed to cover various scenarios, including typical use cases, edge cases, and potential error conditions.

Each unit, whether it is a function, procedure, or module, is tested in isolation from the rest of the system. This isolation ensures that any issues detected are related to the specific unit under examination.

Each test case is self-contained and does not rely on the results of other tests. This independence makes it easier to identify the source of issues when a test case fails.

***Tools***

The pytest module is selected as the primary tool for implementing and organizing unit tests. This module provides a framework for structuring and executing test cases, making it efficient to assess the behavior of individual programs and functions. The unittest module is also used in order to handle mocks/patches used in testing.

Github Actions will be utilized as a Continuous Integration (CI) tool. By implementing CI, code will be automatically tested against all relevant test cases each time a commit is made. By automating integration testing, it will greatly reduce the effort required when merging changes from different developers, as well as the amount of time and resources required for debugging.

***Test Files***

* Collection scripts (excluding scripts without testable functionality; some scripts lack query/mapping builder functions and only act as sequences of function calls from other scripts to retrieve data)
  + test\_bonds\_api\_query.py
  + test\_companies\_api\_query.py
  + test\_historical\_api\_query.py
  + test\_realtime\_api\_query.py
* Database scripts
  + test\_bonds\_insert.py
  + test\_company\_statements\_insert.py
  + test\_historical\_commodity\_insert.py
  + test\_historical\_index\_insert.py
  + test\_historical\_stock\_insert.py
  + test\_obsolete\_data\_deletion.py
  + test\_realtime\_commodity\_insert.py
  + test\_realtime\_index\_insert.py
  + test\_realtime\_stock\_insert.py
  + test\_symbol\_change\_update.py
* Utility scripts
  + test\_api\_handler.py
  + test\_data\_handler.py
  + test\_db\_handler.py
  + test\_file\_handler.py

***Integration Testing***

***Scope***

Integration testing should check the correctness of interactions between components and modules.

***Objective***

The objective of integration testing is to verify that different components interact correctly with each other, and that the interfaces between them function as expected. This testing level ensures that integrated components collaborate to fulfill their intended functions.

***Testing Approach***

The development team will design and execute test cases that focus on the interactions between different modules, components, or services. The integration tests should cover various aspects, including communication between modules, data transit, and the correctness of data transformations and operations.

***Files***

* create\_integration\_db.sql
* create\_integration\_triggers.sql
* integration\_test.sh

***System Testing***

***Scope***

System testing tests the entire system as a whole, representing the top level of the testing process. It evaluates the entire software application or system to ensure it functions as an integrated unit.

***Objective***

The primary objective of system testing is to ensure that the entire system, consisting of all its components and modules, meets the specified requirements. This includes both functional and non-functional requirements. The goal is to validate that the software behaves as expected when all components are interconnected and the system operates as a complete entity.

***Testing Approach***

The development team will evaluate the entire application to determine whether it fulfills its intended purpose, meets user requirements, and operates according to the defined functional and non-functional requirements.

The tests should be approached with a wide range of scenarios, including typical user interactions, edge cases, and potential error conditions.

Test data and environments should closely resemble the production environment, to ensure the software behaves realistically in real-world scenarios.

Non-functional aspects such as performance, security, scalability, and usability will be assessed during system testing.

***Tools***

Bash scripts

cron jobs

pytest and unittest libraries

Git Actions

#### *Test Classes*

##### **Collection Script Test Class**:

**Description**: Test Objective - This class is focused on testing building queries and mappings for specific data collection targets

**Test case:**

* Test that the query builder callback function successfully populates the query manager with valid request strings
* Test that the mapping builder function successfully creates a mapping callback that produces correct non-api field values

##### **Database Script Test Class**

**Description**: This class test the valid insertion of collection output into the database

**Test Cases**:

* Test that the correct entry ID is retrieved from output data
* Test that the correct entry keys are created
* Test that simulated query execution is successful and inserts the right data

##### **Error Handling and Logging Test Class**

**Description**: Verify that the program will log errors and exceptions appropriately

**Test Cases**:

* Log the exceptions that occurred during execution in log file on the server.
* Load this log file into the database logs table
* Verify that logs contain relevant information for troubleshooting.
* Log messages will follow 5 standard log levels to distinguish them : *CRITICAL, ERROR, WARNING, INFO, AND DEBUG*
* System logs can be viewed via phpMyAdmin, the ATS User Interface, or directly through the servers file directory on IONOS

#### General Test Conditions

Testing will be conducted on a dedicated AlmaLinux server hosted by Okanagan College. When features, hotfixes, or other merges are made they will be tested on this server prior to being pushed onto the production server (IONOS). During collection, static test data files will be used to verify incoming data, and ensure it is formatted as expected by the database

### *Planned Testing*

* Database integration
* API integration
* Data collection
* Data storage
* Obsolete data deletion
* Error handling
* Interactions between modules

#### *Module Test*

All of the modules to be tested (refer to the Software Requirements Traceability Matrix) are to be tested using defined module level test methodology. For details regarding test procedures and setup. The resulting outputs of this test are Internal Test Reports (ITR) or Unit Test Report (UTR). When all of the module tests for a module are passed the module is ready for integration level testing.

## Test Schedules

The following test will be run automatically through continuous integration as features are committed.

Unit Testing: Automatically when code is committed

* Objective: Ensure each component of the system is correct.
* Deliverable: Functional code block for parts of a component.

Integration Testing: Automatically as branches are merged

* Objective: Data collection and the database function as expected after the combination of components.
* Deliverable: Components of the system that changed log reports and pass the integration tests

System Testing: Prior to acceptance testing

* Objective: Ensuring that the system is functioning normally, and the baseline is error free after the integration tests pass.
* Deliverable: System tests passed and test metrics recorded and logged.

Acceptance Testing: Before the end of every iteration

* Objective: Get client feedback on the system issues, functionality, and requirements satisfaction.
* Deliverable: System demonstration and client approval.

Release Testing: Before every release of the product

* Objective: Conduct performance, scenario testing on the system to best optimize the system in order to satisfy the needs before final release.
* Deliverable: A stable and applicable system meeting the requirements of the clients

## Risk Management

### Third-Party Dependencies

The system relies on multiple services and/or tools provided by third-party partners. Disruptions or changes to these third-parties could have significant impact on the System, and by extension the Testing Plan. These third-parties include, but are not limited to:

* Financial API service
* IONOS web hosting services
  + including: Database & Database Management, Warehouse & Warehouse Management, Server Tools for execution of scripts
* GitHub Repository
* Github Actions
* Atlassian Products
  + Jira
  + Confluence

#### CONTINGENCY PLAN(S)

The activation of third-party technical support will be required for all scenarios where this is an appropriate option. Submission of tickets to third-party support teams will be done by the Product Owners or system administrator.

2. In the event of API service outage or disruption, there will be significant disruption to the system.. To deal with this scenario, the system will include support for alternative RESTful API services, so that in the case of an outage an alternative data source may be used as a replacement.
4. In the event that there is an outage or disruption of the IONOS web hosting services, on-premise servers at Okanagan College will be utilized as a contingency. Product Owners will maintain access to these servers, and provide access to Development Team members in the event of this contingency plan being activated. The system will be redeployed on these servers utilizing the most recent release or iteration.
6. In the event that there is an outage or disruption of GitHub services, teams will continue to develop feature branches based on their local copy of the repository. An on-premise development server will be leveraged to perform daily backups of the GitHub codebase to ensure continuity in the event of prolonged outage.

### In the event of outage or disruption of Atlassian products and services, multiple contingency plans may be enacted depending on which service is inaccessible. In the event of Confluence outages or disruptions, Google Drive will be utilized as a contingency. Documentation updates, changes, etc. will be managed on Google Drive until access to Confluence resumes. In the event of Jira outages or disruptions, Discord or Zoom will be leveraged to ensure continuity of task delegation and communication of deadlines or expectations. Team members will track time spent on their assigned tasks via separate means, and update tickets accordingly once access to the service is restored.

### System Requirements

The system is a software product leveraging Agile methodologies for development. As the system is iterated on and delivered to the client in increments, changes may be necessary to align with the client’s expectations. The Testing Plan assumes changes to the system will fit within the current testing methods.

#### CONTINGENCY PLAN(S)

To mitigate risks posed by changes, all changes post-acceptance of initial requirements will follow the Change Request Policy [R6], as outlined in the Software Configuration Plan [R5]. This will allow the Product Owners to assess and determine the risks posed by changes that are requested.

### Availability of Resources

The Testing Plan assumes that certain resources will be available for testing. Resources include but are not limited to:

* Development Team
* Client(s)
* Lab Rooms at Okanagan College Kelowna Campus

#### CONTINGENCY PLAN(S)

All efforts are made to ensure consistent communication of availability of the Development Team and their current tasks. In the event of resource requirements exceeding the availability of the Development Team, the Product Owners will meet with the Development Team and Client to discuss revising the scope of the current sprint.

In the event of lab rooms being unavailable, meetings and activities will be coordinated via Zoom or Discord, as appropriate.

### Alignment of Development and Testing Environments

The Testing Plan assumes testing will occur in a dedicated testing environment before updates or iterations can be deployed to the acceptance environment. The testing environment will be held on a private college server, configured to replicate the production environment as closely as possible. System issues may exist in one environment but not the other.

#### CONTINGENCY PLAN(S)

In the case of test servers being down, a system release may need to be delayed until testing can occur. A system administrator will need to be contacted to solve these issues. This is to ensure that production features are up to maximum quality and consistency. If an event occurs where testing passes on the test server, but fails when reaching production, the system will be rolled back to the last-working-version while these inconsistencies are resolved.

## Requirements Traceability

For details regarding the Requirements Traceability Matrix, refer to the separate Requirements Traceability Matrix document [R3]