Software Configuration Management Plan (SCMP)
Stock Trend Analysis Dashboard (STAD)

1 Introduction

This document describes the software configuration management activities to be performed in support of the Stock Trend Analysis Dashboard (STAD) project. The STAD project is charged with developing an application that takes in a CSV file with historical stock data and generates an interactive dashboard for analyzing investment strategies.

1.1 Purpose

The software configuration management plan (SCMP) for the STAD project describes how the project team's software development practices will aid in the efficient development of the data pipeline and dashboard that will constitute the resulting application.

1.2 Scope

This document defines the SCM activities necessary to develop, document, and test the code base for the STAD project. This includes a backend system for data validation, preprocessing, and machine learning integration, as well as a frontend dashboard application developed using React and Vite. The frontend application provides user interaction functionality, dataset upload capabilities, configuration of analysis parameters, and visualization of processed stock market data through interactive charts and dashboard components.

1.3 Organizational Relationships

The STAD project shall be completed as part of the Software Engineering course (CS 673) at Boston University's Metropolitan College. In order to fulfil the course's project requirements, the following deliverables will be submitted by their corresponding deadlines:

Project Proposal 02/03/2026

Software Configuration Management Plan 02/10/2026

Software Project Management Plan 02/17/2026

Software Requirements Specification 02/24/2026

Mid-semester Presentation 03/03/2026

Software Design Document 03/24/2026

Final Presentation 04/28/2026

Final Iteration of the Repository 04/28/2026

The final iteration of the project repository shall comprise the project source code, tests, and documentation.

The frontend dashboard source code is maintained in the following repository:
`https://github.com/kuanysh100322/stock-trends-analysis-dashboard-frontend`

The backend application source code is maintained in the following repository:
`https://github.com/atharvaDS05/Backend-Structure`

1.4 Definitions and Abbreviations

1.4.1 Abbreviations

- SCM: Software Configuration Management
- SCMP: Software Configuration Management Plan
- STAD: Stock Trend Analysis Dashboard
- CI: Configuration Item
- DRF: Django REST Framework
- OHLCV: Open, High, Low, Close, Volume
- SCR: Software Change Request
- CCB: Configuration Control Board

1.5 References

- ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management
- IEEE Std 828-1998, IEEE Standard for Software Configuration Management Plans
- IEEE Std 828-2012, IEEE Standard for Configuration Management in Systems and Software Engineering

2 SCM Management

2.1 Organization and Responsibilities

The responsibilities of the STAD project shall be divided into five roles, defined below. The latter four shall report to the project manager, who is directly responsible for the project development.

2.1.1 Project Manager

The project manager is responsible for planning the general structure of the project, coordinating the project timeline, organizing team meetings, tracking the completion of project tasks, documenting the project's progress, and presenting the results of the project.

2.1.2 Dashboard/UI Developer

The Dashboard/UI Developer is responsible for designing, implementing, and maintaining the frontend dashboard application using React and Vite. Responsibilities include developing reusable UI components, implementing navigation and layout structures, and integrating frontend components with backend REST APIs.

The Dashboard/UI Developer ensures that user interactions such as dataset uploads, filtering options, and visualization controls correctly trigger backend processing and update dashboard displays. The developer is also responsible for maintaining responsive design, frontend performance optimization, usability, and accessibility requirements.

Additionally, the Dashboard/UI Developer participates in frontend testing, debugging, and documentation of UI workflows and component functionality.

2.1.3 Data/Backend Developer

The data/backend developer is responsible for designing and implementing the backend services for the project using Django and the Django REST framework. This includes but is not limited to API development; creating and managing the database and data processing pipelines; design and implementation of business logic; and development of user authentication practices.

2.1.4 ML/QA Engineer

The ML/QA engineer is responsible for the design, implementation, validation, and testing of the machine learning models that will be used to analyze the stock data. These models will be implemented in Python using relevant Python libraries such as scikit-learn and its associated modules.

2.1.5 QA and Documentation Support Engineer

The QA and documentation support engineer is responsible for contributing to the above three areas of the project (dashboard/UI, data/backend, and ML) and for maintaining the integration between them. Furthermore, they are responsible for maintaining clear documentation and developing tests of the code base, including integration testing between frontend and backend sections.

2.2 Team Members and Contributions

The STAD project is developed by a collaborative team of students, each responsible for specific system components and development activities. The distribution of responsibilities ensures balanced participation across frontend development, backend implementation, machine learning integration, testing, and project management.

- **Ulzhalgas Seidaliyeva – Project Manager** Ulzhalgas Seidaliyeva serves as the Project Manager and is primarily responsible for coordinating the overall project structure, organizing team meetings, tracking project milestones, managing deliverables, and maintaining project documentation. In addition to management responsibilities, Ulzhalgas contributes to dashboard development, backend implementation support, and quality assurance activities.
- **Kuanysh Amandos – Dashboard/UI Developer and Data/Backend Developer** Kuanysh Amandos serves as the primary developer for frontend dashboard implementation

using React and Vite. Responsibilities include designing reusable UI components, implementing data visualization features, integrating frontend components with backend APIs, and maintaining frontend performance and usability. Kuanysh also contributes significantly to backend development, including API integration and system architecture support.

- **Haoqian Zhang – Dashboard/UI Developer and Data/Backend Developer** Haoqian Zhang contributes primarily to frontend dashboard development and backend system implementation. Responsibilities include assisting in user interface development, supporting backend service implementation, and ensuring integration between frontend and backend components.

- **Atharva Sharma – ML/QA Engineer** Atharva Sharma serves as the primary ML/QA Engineer and is responsible for implementing machine learning modules, validating model performance, developing testing procedures, and ensuring system reliability. Atharva also contributes to dashboard and backend development activities as needed.

- **Mackenzie Kong-Sivert – QA and Documentation Support** Mackenzie Kong-Sivert contributes to quality assurance and project documentation activities. Responsibilities include assisting with testing procedures, validating system functionality, supporting documentation updates, and participating in requirements clarification and usability evaluation. Mackenzie also supports integration testing between frontend and backend components.

2.3 Interface Control

Interface control ensures consistency between frontend, backend, and machine learning components. The STAD system uses REST APIs to support communication between system modules. API request and response formats are documented and maintained to ensure stable integration. Changes to interface specifications are managed through version control processes.

2.4 SCMP Implementation

The SCMP is implemented through Git-based version control and repository management using GitHub. Configuration management activities include version tracking, change control, documentation updates, and repository organization. Team members follow structured commit and review workflows to maintain configuration consistency throughout the development lifecycle.

2.5 Applicable Policies, Directives, and Procedures

The STAD project follows standard software engineering and configuration management practices, including structured repository organization, descriptive commit messages, peer review of code changes, and documentation maintenance standards. These practices ensure quality, traceability, and maintainability of project artifacts.

3 SCM Activities

3.1 Configuration Identification

Configuration identification defines the software components, data artifacts, documentation, and supporting materials that are subject to configuration management throughout the lifecycle of the

Stock Trend Analysis Dashboard (STAD) project. All configuration items are maintained within the project's GitHub repository to ensure proper version control, traceability, and reproducibility.

Configuration items include:

- backend source code (Django, DRF)
- frontend source code (React, Vite)
- machine learning modules
- documentation
- test data
- data artifacts

Beyond high-level categories, the STAD project identifies the following specific Backend Configuration Items (CIs):

- **Backend Application Source Code:** API layer (Django + DRF) including routing, serializers, and middleware. Service layer business logic: *market_data_service* (yfinance integration), *feature_engineering_service* (indicators), *labeling_service* (invest/no-invest generation), *backtesting_service*, and *ml_service*.
- **Configuration / Runtime Assets:** Dependency manifests (`requirements.txt`), runtime templates (`.env.example`), and Dockerfiles.
- **Data Contracts and Schemas:** Internal canonical time-series schemas (date, open, high, low, close, volume). API contract documentation (OpenAPI/Markdown).
- **Persistence Artifacts:** Django database migrations and local cache configurations to reduce repeated ticker downloads.
- **Operational Guardrails:** Retry/backoff rules for upstream fetch failures, Cache TTL policies, and rate limiting definitions.

3.2 Configuration Control

Configuration control ensures that all changes to configuration items are properly reviewed, approved, and documented to maintain system integrity and consistency. The STAD project uses Git and GitHub as the primary tools for configuration control.

All modifications to configuration items must be committed to the repository with clear and descriptive commit messages. These commit messages must describe the nature of the change, the purpose of the modification, and the components affected.

The project distinguishes between change types to determine the level of oversight required:

- **Minor Changes (PR review only):** Bug fixes, improved logging, refactoring without behavior change, or adding non-breaking response fields.
- **Major Changes (SCR + CCB required):** Changes to the canonical data schema, labeling rules/thresholds, breaking API changes, or changing the upstream fetch strategy.
- **Version Rules:** Breaking changes require a version increment (e.g., /api/v1/ to /api/v2/) and updated contract docs.

3.2.1 Controlled API Interfaces

The following backend services and endpoints are under configuration control:

1. **Run Initialization:** POST /api/v1/runs (Ticker → Data Fetch)
2. **OHLCV Retrieval:** GET /api/v1/runs/{id}/ohlcv
3. **Indicator Computation:** POST /api/v1/runs/{id}/indicators
4. **Label Generation:** POST /api/v1/runs/{id}/labels
5. **Backtesting:** POST /api/v1/runs/{id}/backtest
6. **Summary KPIs:** GET /api/v1/runs/{id}/summary

3.3 ML Artifact & Lineage Control

To ensure reproducibility, ML artifacts follow a "registry-lite" versioning policy:

- **ML CI Inventory:** Serialized model objects (joblib/pickle), preprocessing pipeline artifacts (scalers/encoders), and evaluation artifacts (metrics.json, confusion matrices).
- **Lineage Tracking:** Every model run is stored with a `run_metadata.json` containing the ticker, date range, dataset fingerprint (hash), Git commit SHA, and random seed.
- **Storage Pattern:** Artifacts are stored in an append-only structure: ml_artifacts/ticker/interval/model/run_id/.

3.4 Configuration Status Accounting

Configuration status accounting provides visibility into the current and historical status of configuration items. GitHub maintains a complete history of all changes, including the identity of the contributor and the time of the change. The repository serves as the authoritative source for the current version of all configuration items.

3.5 Testing & Quality Assurance

The SCM process enforces a test pyramid as a release gate:

- **Unit Tests:** Mandatory for data normalization, feature functions, labeling logic, and API serializers.
- **Integration Tests:** Validates the full "Ticker → Fetch → Label → Backtest" pipeline and cache behavior.
- **Data Quality Checks:** Validates for missing values, non-monotonic dates, and unreasonable prices (negative values) before processing.
- **Contract Tests:** Ensures API responses contain all required fields used by the React UI.

3.6 Audits and Reviews

Configuration audits and reviews ensure configuration items are properly maintained. Team members follow a "Definition of Done" for PRs:

- PR reviewed and approved by at least one peer.
- All unit and relevant integration tests pass.
- Contract documentation updated if the response schema changed.
- SCR issue updated with the implementation details if it was a major change.

4 Tools, Techniques, and Methodologies

The STAD project uses established software development tools. Git and GitHub serve as the central repository. The backend uses Python, Django, and Django REST Framework. Data processing is handled via Pandas and NumPy. The frontend is developed using React and Vite, with modern charting libraries for visualization.

To protect system integrity when interacting with external providers (yfinance), retry/backoff rules and cache policies are under configuration control.

5 Records Collection and Retention

The dashboard's source code, test data, and documentation shall be stored within project GitHub repositories. Test procedures and data resulting from the successful tests shall be retained. Final iteration of the repository is due on 04/28/2026.