Stock Portfolio Management System: A Model-Driven Agile Approach Using Spiral Methodology and UML

Krish Gada
Student, Department of Computer Engineering
SVKM NMIMS MPSTME
Mumbai, India
krishgada07@gmail.com

***Abstract* - This paper presents the comprehensive design of a Stock Portfolio Management System (SPMS) utilizing a hybrid approach that integrates the Spiral Development Model with Agile Scrum methodology. The proposed system aims to enable users to monitor investments, access real-time market data, and receive personalized insights based on individual risk profiles. While the system has not been implemented, its architecture and design have been meticulously developed using System Analysis and Design (SAD) principles. Unified Modeling Language (UML) diagrams—including Use Case, Class, Sequence, Activity, and State diagrams—are employed to represent various facets of the system. The Spiral Model facilitated our iterative, risk-conscious planning, while Scrum structured our workflow into incremental sprints. Our contribution lies in applying these methodologies to a domain where user-centric design and technical robustness will be equally critical, providing a detailed blueprint for future implementation.**

Keywords— Agile Scrum, Spiral Model, UML, Portfolio Management, Software Engineering, System Analysis and Design.

I. INTRODUCTION

In today's volatile financial markets, effective investment management requires more than just access to data; it necessitates tools that can contextualize information, support informed decision-making, and adapt to individual investor goals. While numerous applications exist for tracking stocks, few offer a system that balances functionality, personalization, and thoughtful design. This project addresses that gap by proposing a Stock Portfolio Management System (SPMS) that is both theoretically sound and practically applicable.

Our approach integrates two key methodologies: the Spiral Model and Agile Scrum. The Spiral Model assists in planning through iterative cycles, evaluating risks, and refining concepts continuously. Concurrently, Scrum breaks down the work into manageable sprints, fostering team collaboration and adaptability. Our focus has been on detailed system design rather than implementation.

We have employed System Analysis and Design (SAD) principles supported by Unified Modeling Language (UML) diagrams to shape our design process. Each diagram captures a specific aspect of the system—from user interactions to system behavior and data flow. The outcome is a set of models that collectively present a comprehensive plan for building a responsive and intelligent investment management tool. This paper details how

each methodology informed our process and how the system was envisioned through design artifacts.

II. LITERATURE REVIEW

Designing complex, user-centric systems in volatile domains like finance necessitates robust methodologies and modeling frameworks. Over the past two decades, System Analysis and Design (SAD) practices have increasingly shifted toward model-driven development using Unified Modeling Language (UML), supported by iterative and adaptive methodologies like Agile Scrum and the Spiral Model. This literature review contextualizes our work by exploring foundational studies and contemporary advancements in each of these areas.

A. Spiral Model in Software Engineering

Barry Boehm's Spiral Model introduced a paradigm that combines iterative development with risk management, allowing design teams to progressively refine systems through loops of planning, risk analysis, and evaluation [1]. The model has been widely adopted in high-stakes environments such as defense, finance, and aerospace, where system failure carries significant consequences. Research has shown its strength in accommodating changing requirements and its suitability for large-scale or exploratory projects. Our adoption of the Spiral Model reflects these advantages, particularly in addressing the unpredictable nature of stock market logic and evolving user needs in portfolio management.

B. Agile Scrum for Iterative Workflow

Agile methods emerged in response to the limitations of rigid, plan-driven approaches. Scrum, as formalized by Schwaber and Sutherland, provides a lightweight framework that promotes iterative development, team collaboration, and customer involvement through sprints, backlogs, and reviews [2]. Although originally conceived for code-intensive projects, Agile principles have been successfully extended to system design phases. Studies such as those by Ambler have emphasized Agile Modeling as a complement to documentation-heavy processes, enabling the creation of just-enough design artifacts that evolve with the system. In our case, Scrum was applied as a structuring mechanism to organize UML-based deliverables within timed design cycles, even without actual code implementation.

C. Unified Modeling Language (UML)

UML has become the de facto standard for representing software systems visually. Each UML diagram type plays a unique role in conveying structure, behavior, and interaction. Use Case diagrams are effective in capturing user goals and functional requirements, serving as early alignment tools between developers and stakeholders [3]. Class diagrams offer a blueprint of system architecture, showing attributes, operations, and relationships between objects. Sequence diagrams, by contrast, illuminate the flow of messages between entities over time, aiding in behavior modeling and process comprehension [5].

Activity diagrams and state diagrams have been used to map out workflows and system reactivity. Cysneiros and Leite explored how non-functional requirements can be transformed into conceptual behavioral models, highlighting the utility of these diagrams in robust system design [7].

D. Requirements Engineering with UML

UML's role in requirements engineering has been well-documented. Saeed et al. demonstrated how UML supports efficient requirement gathering and validation in small-scale projects, showing that its adaptability and visual expressiveness facilitate early defect detection and better stakeholder communication [6]. Additionally, IEEE Std 830-1998 formalizes the role of structured requirements in software lifecycle documentation, offering guidelines that reinforce the use of modeling as a form of traceable specification [4].

E. Behavioral Finance and Personalization in FinTech

Our project also draws conceptual influence from the field of behavioral finance. As argued by Thaler, investors often act irrationally and are influenced by biases and heuristics [8]. Designing a system like SPMS demands that such behavioral factors be incorporated into the insight generation process. Current literature underscores the inadequacy of data-only platforms and advocates for systems that personalize financial guidance using psychological

segmentation and risk modeling. Our InsightEngine design is informed by this gap, offering an academically grounded enhancement over existing platforms.

F. Gaps in Existing Systems and Research

Most current stock portfolio applications emphasize data aggregation rather than intelligent design or architecture transparency. Few publicly accessible case studies exist where modeling, documentation, and methodology are clearly aligned. Our work addresses this gap by providing a fully modeled, methodology-backed design blueprint for a financial system—demonstrating how SAD, Agile, and UML can be effectively combined for intelligent FinTech applications.

III. METHODOLOGY

The methodology we adopted combines the structural depth of the Spiral Model with the flexibility of Agile Scrum. This hybrid approach was essential not only for managing the scope of the system but also for aligning our design with real-world development challenges.

A. Spiral Model

The Spiral Model, introduced by Boehm in 1986, is a risk-driven process model that combines iterative development with systematic aspects of the traditional waterfall model. It consists of four primary phases: Planning, Risk Analysis, Engineering, and Evaluation. Each iteration, or spiral, involves these phases, allowing for incremental refinement of the system. This model is particularly suited for complex projects with evolving requirements, as it emphasizes risk assessment and mitigation throughout the development process [1].

In our project, the Spiral Model provided a foundation to think through the system in a structured yet iterative manner. We began with the identification of objectives, constraints, and possible risks associated with building a stock portfolio management system. Each phase of the spiral included planning, risk analysis, modeling, and evaluation. This approach allowed us to explore ideas without prematurely locking ourselves into design decisions. At each loop of the spiral, we re-examined our assumptions and refined the design artifacts accordingly. This was especially useful when considering complex system behaviors, such as automated insight generation or portfolio rebalancing based on user-defined risk profiles.

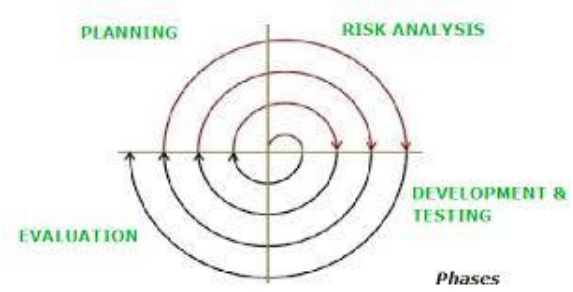

Figure 1: Spiral Diagram

B. Agile Scrum

Agile methodologies, including Scrum, emphasize iterative development, collaboration, and adaptability. Scrum structures the development process into time-boxed iterations and sprints, typically lasting two to four weeks. Each sprint results in a potentially shippable product increment, with progress reviewed in sprint reviews and processes evaluated in sprint retrospectives [2].

In our project, we structured our design tasks using Agile Scrum. Although we did not code or build the system, we simulate the development process through sprints. Each sprint began with a planning phase where specific goals were identified, such as defining user roles, drafting the use case model, or elaborating the class diagram. Daily stand-up meetings allowed us to communicate progress and challenges, even in a non-technical design-focused setting. By the end of each sprint, we reviewed our outputs—often in the form of updated diagrams or revised documentation—and held retrospectives to evaluate what worked well and what needed adjustment.

We found that Scrum complemented the Spiral Model effectively. While the Spiral encouraged long-term thinking and systematic iteration, Scrum helped us manage short-term goals and maintain momentum. The two approaches together created a

rhythm that was both disciplined and adaptive. More importantly, this combination helped us envision how a real development team might approach building such a system and ensured that our designs were grounded in realistic processes.

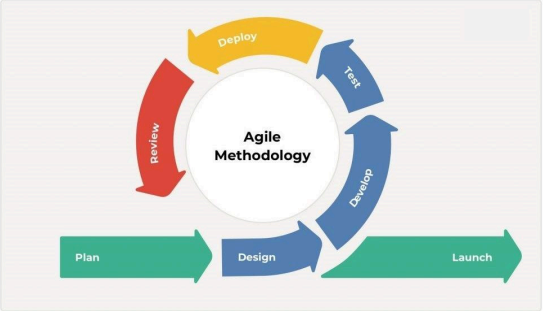

Figure 2: Agile Scrum

IV. REQUIREMENTS OF ENGINEERING

A significant phase in our design process involved identifying and refining both functional and non-functional requirements.

A. Functional Requirements

Functional requirements outlined user-facing capabilities, including:

- User Registration and Authentication**:** Users must be able to create accounts and log in securely.
- Portfolio Management**:** Users can add, remove, and modify assets in their portfolio.
- Real-Time Market Data Access**:** The system should provide up-to-date market information.
- Insight Generation: Personalized investment insights based on user risk profiles.

Each requirement was reviewed for completeness, and scenarios were developed to test potential implementations.

B. Non-Functional Requirements

Non-functional requirements were equally emphasized, focusing on:

- Performance: Market data must refresh quickly to provide real-time information.
- Security: Sensitive user information must be stored securely, with robust authentication mechanisms.
- Reliability: The system should have high availability and handle errors gracefully.
- Usability: The interface should be intuitive and user-friendly to cater to investors with varying levels of expertise.

These requirements shaped the features and influenced choices in UML modeling.

V. SYSTEM DESIGN AND UML MODELING

Unified Modeling Language (UML) provides a standardized way to visualize system design. We utilized various UML diagrams to represent different aspects of the SPMS.

A. Use Case Diagram

Our first step in understanding the system's functionality was to construct a Use Case diagram. This diagram allowed us to explore what features the system needed to provide from the user's perspective. The primary actor is the investor, who interacts with the system to perform actions such as logging in, managing their portfolio, viewing real-time market updates, and receiving investment insights. The diagram served as a foundation for our functional requirement gathering and later guided the creation of more detailed models. Its importance lies in showing how external users view the system's services, making it essential for aligning development with user expectations [3].

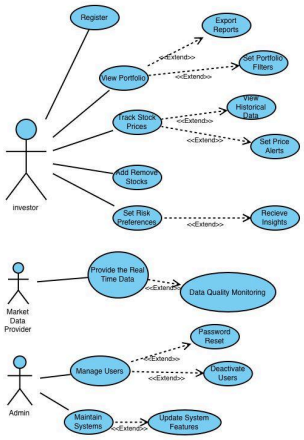

Figure 3: Use Case

B. Class Diagram

The Class diagram was used to lay out the structural relationships and core entities of our system. The key classes we identified include User, Portfolio, Stock, Holding, MarketDataAPI, InsightEngine, and ReportGenerator. Each class was annotated with its attributes and methods to show how data and functionality were encapsulated. For example, the Portfolio class holds a list of Holding objects, which link to specific stocks and quantities. The InsightEngine class takes a user's portfolio and risk profile as inputs to produce tailored recommendations. The class diagram brought conceptual clarity to the system and established a strong foundation for future implementation and scalability [4].

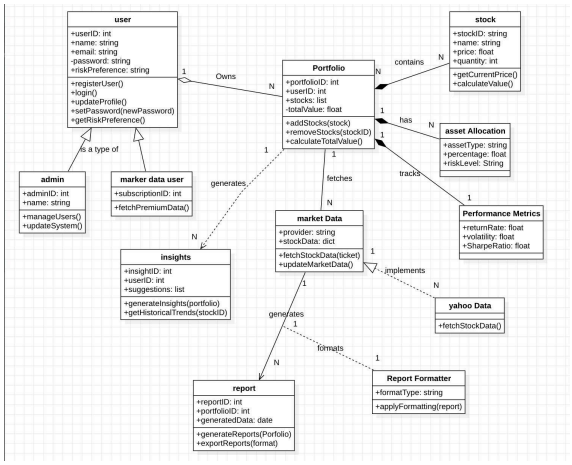

Figure 4: Class Diagram

C. Sequence Diagram

To depict the temporal interaction between system components, we developed a Sequence diagram centered around the generation of investment insights. In this use case, a user initiates a report request, which triggers a sequence of method calls starting from the UI layer to the PortfolioManager, InsightEngine, and ReportFormatter. The diagram highlighted how objects communicate in a time-ordered manner and validated the logical sequence of operations, data flows, and method calls. This visualization ensured that class interfaces were cohesively aligned and that each subsystem contribute meaningfully to end-user functionality [5].

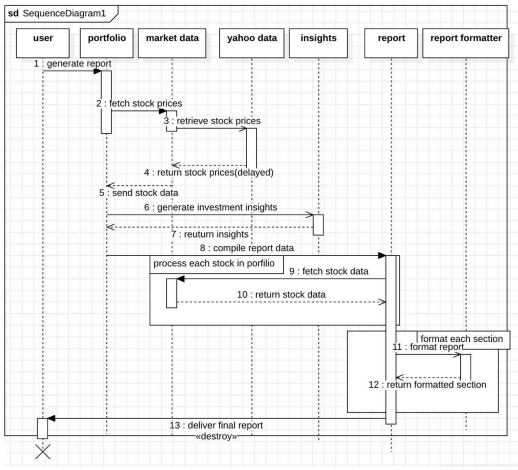

Figure 5: Sequence Diagram

D. Activity Diagram

The Activity diagram modeled the user authentication process. This flowchart-like diagram illustrates decision points such as whether a user is new or returning, login credential submission, validation, and system responses (e.g., successful login or authentication failure). Incorporating edge cases like invalid passwords or server issues allowed us to better understand and design system resilience. Moreover, the diagram underscored the importance of UX-friendly error handling and redirect mechanisms, which are crucial for user retention and trust [6].

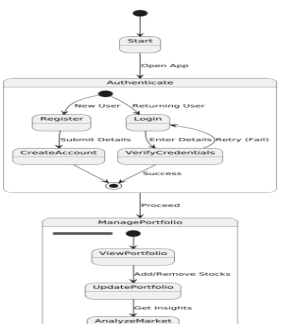

Figure 6: Activity Diagram

E. State Diagram

Our State diagram focused on the behavioral changes in the Portfolio object, specifically regarding the investment insight lifecycle. The portfolio transitions between states such as "Insights Outdated", "Insights Requested", "Insights In Progress", and "Insights Up-to-date", based on user actions and market data changes. This model helped us articulate how the system should automatically respond to changes, such as notifying the user to update insights when the market shifts significantly or when portfolio allocations are altered. Behavioral modeling is particularly crucial in dynamic domains like finance, where real-time data necessitates adaptive system responses [7].

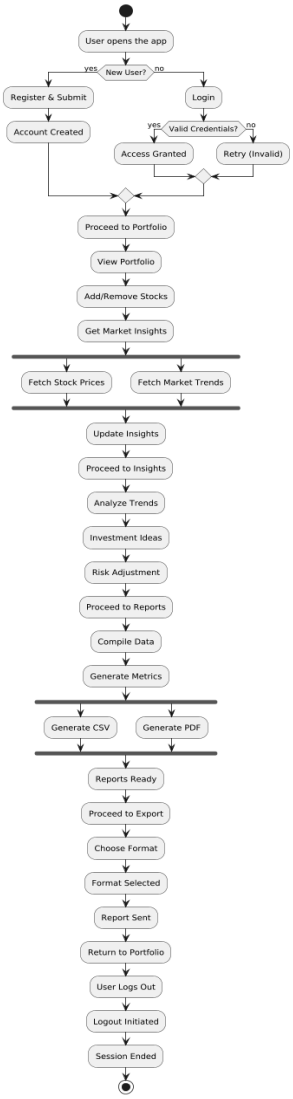

Figure 7: State Diagram

VI. COMPARATIVE ANALYSIS

Existing stock portfolio applications like Zerodha Console, Groww, and Yahoo Finance are primarily data-driven, offering limited personalization or predictive insights. Most platforms focus on metrics display rather than decision support based on the user's investment psychology or risk appetite.

Our proposed SPMS distinguishes itself through the integration of a personalized InsightEngine, which uses user-defined risk profiles to recommend asset reallocation, suggest diversification strategies, and alert users of high-volatility assets. Unlike existing systems that treat users uniformly, our design is based on segmentation by investor behavior—aligning with modern behavioral finance principles [8].

Furthermore, the majority of existing systems lack transparency in architectural documentation or traceability from design to implementation. By employing model-driven design with UML, our project ensures that each requirement is systematically traced to corresponding diagrams, thereby supporting better communication across technical and non-technical stakeholders.

VII. DISCUSSION

The development of this system provided insights into the synergistic use of Spiral methodology and Agile Scrum for software design without implementation. These methodologies are not mutually exclusive but can be combined to offer both structure and flexibility. The Spiral Model guided us in managing risks and planning iterative refinements, while Scrum provided a rhythm to execute tasks within time-boxed iterations.

Each UML artifact contributed uniquely:

- Use Case Diagrams anchored development in user needs.
- Class Diagrams clarified relationships among key entities.
- Sequence Diagrams illustrated process flows and dependencies.
- Activity Diagrams mapped possible user paths and system reactions.
- State Diagrams ensured robust handling of dynamic behaviors.

Together, these diagrams not only served as documentation tools but also as validation mechanisms. They allowed us to uncover potential design flaws early and adapt the design iteratively—replicating a real-world development environment despite the absence of actual code.

Moreover, the SPMS project reinforced the value of model-driven architecture (MDA) in high-stakes applications like financial tools. By modeling before coding, one can ensure alignment with stakeholder goals, minimize rework, and lay a clear foundation for future development.

VIII. CONCLUSION

Through this research, we have developed a detailed, model-driven design for a Stock Portfolio Management System that combines user-centered design with strong engineering principles. The use of the Spiral Model allowed us to account for evolving system risks and requirements, while Agile Scrum ensured incremental progress and team agility.

Our comprehensive UML modeling approach ensured that abstract requirements were transformed into actionable designs, creating a development-ready blueprint. Even though the system has not been implemented, it serves as a case study for how SAD and Agile practices can work in tandem to envision complex systems.

We believe our contribution extends beyond just design—it offers a replicable methodology for designing intelligent, user-centric systems in financial technology. Future work may involve implementing the design with real-world APIs (e.g., Alpha Vantage or NSE APIs), integrating machine learning models for insight generation, and conducting user acceptance testing (UAT) with real investors to refine usability and feature relevance.

REFERENCES

[1] B. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14–24, 1986.

[2] K. Schwaber and J. Sutherland, "The Scrum Guide," Scrum.org, 2020. [Online]. Available: https://scrumguides.org

[3] M. Ražinskas et al., “Transforming Sketches of UML Use Case Diagrams to Models,” *IEEE Access*, vol. 12, pp. 185826–185839, 2024.

[4] IEEE Computer Society, *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998, 1998.

[5] S. Abrahão et al., “Assessing the Effectiveness of Sequence Diagrams in the Comprehension of Functional Requirements,” *IEEE Trans. Software Eng.*, vol. 39, no. 3, pp. 327–342, Mar. 2013.

[6] M. S. Saeed et al., “Efficient Requirement Engineering for Small Scale Projects Using UML,” in *Proc. IEEE ICCSA*, 2016, pp. 660–667.

[7] L. M. Cysneiros and J. C. S. do Prado Leite, “Nonfunctional Requirements: From Elicitation to Conceptual Models,” *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 328–350, May 2004.

[8] R. Thaler, “Behavioral Finance: Past Battles and Future Engagements,” *Financial Analysts Journal*, vol. 55, no. 6, pp. 12–17, 1999.