ST JOSEPH ENGINEERING COLLEGE, VAMANJOOR, MANGALURU

SOFTWARE REQUIREMENT SPECIFICATION

on

CRICNEX – MACHINE LEARNING PLAYER ANALYTICS FRAMEWORK

By

PRAJWAL J

4SO24MC065

Department of Computer Applications

St Joseph Engineering College

Mangaluru-575028

2025

3.1 INTRODUCTION

3.1.1 Purpose

The purpose of this Software Requirement Specification (SRS) is to describe the detailed requirements for the software project titled **CricNex – Machine Learning Player Analytics Framework**.

This document defines the functional and non-functional requirements of the system, the intended users, and the overall product behavior. It serves as a foundation for design, development, testing, and maintenance.

The CricNex system aims to develop a machine learning-based framework capable of predicting cricket player performance metrics such as runs scored, wickets taken, and strike rate by analyzing historical match data. The system integrates multiple predictive algorithms including Random Forest, XGBoost, ARIMA, and LSTM to capture both static and time-series patterns in player performance.

The SRS will help developers, testers, faculty evaluators, and stakeholders understand the system's purpose, operational boundaries, and intended outcomes. It also serves as a contractual agreement for what the system will deliver upon completion.

3.1.2 Document Abbreviations and Definitions

To ensure clarity and uniform understanding, the following abbreviations and definitions are used throughout this document:

- **ML:** Machine Learning
- **AI:** Artificial Intelligence
- **API:** Application Programming Interface
- **DB:** Database
- **UI:** User Interface
- **KPI:** Key Performance Indicator
- **ARIMA:** AutoRegressive Integrated Moving Average (statistical forecasting model)
- **LSTM:** Long Short-Term Memory (deep learning time-series model)
- **RMSE:** Root Mean Square Error (model evaluation metric)
- **MAE:** Mean Absolute Error (model evaluation metric)
- **R^2:** Coefficient of Determination
- **IPL:** Indian Premier League
- **ODI:** One Day International
- **CSV:** Comma-Separated Values (dataset format)

3.1.3 Target Audience

The target audience of this SRS includes:

- Project developers and programmers responsible for implementation
- Testing and QA teams who will validate system performance
- Project guides and academic evaluators who assess compliance with software engineering standards
- End users such as cricket coaches, selectors, analysts, and sports data scientists who will use the system for prediction and analysis
- Researchers and students working in the domain of sports analytics or predictive modeling

3.1.4 Project Scope

The CricNex system provides an intelligent platform for predicting cricket player performance using historical data, contextual factors, and machine learning models.

The scope of the project includes:

- Collecting and cleaning large-scale cricket datasets from sources like Kaggle
- Performing data preprocessing and feature engineering to improve model accuracy
- Developing multiple predictive models (Random Forest, XGBoost, ARIMA, LSTM)
- Comparing the models using metrics such as MAE, RMSE, and R^2 Score
- Building a full-stack web application using Flask (backend) and React.js (frontend)
- Integrating MongoDB for storing player data, match statistics, and predictions
- Designing an interactive dashboard for visualization and comparative analysis

Out of scope:

- Real-time live match prediction
- Video-based performance analytics
- Sentiment analysis or injury prediction
- Prediction of entire team outcomes (focus is on individual player performance only)

3.1.5 Benefits

CricNex provides multiple benefits over traditional manual evaluation methods:

- Enables data-driven player evaluation rather than subjective assessment
- Identifies hidden performance trends and contextual strengths
- Improves accuracy in team selection and player strategy planning

- Reduces the time spent by analysts on manual calculations
- Offers a scalable platform for future research in sports analytics
- Delivers visual, easy-to-understand predictions and comparative charts

3.1.6 References

1. Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning*, Springer
3. Pankaj Jalote, *An Integrated Approach to Software Engineering*, Narosa Publishing House
4. www.kaggle.com – Cricket datasets repository
5. www.analyticsvidhya.com – Machine learning resources and tutorials
6. https://flask.palletsprojects.com – Flask official documentation
7. https://react.dev – React.js documentation
8. https://docs.mongodb.com – MongoDB documentation
9. https://xgboost.readthedocs.io – XGBoost library documentation
10. https://scikit-learn.org – Scikit-learn ML library documentation

3.2 OVERALL DESCRIPTION

3.2.1 Identification of Pre-Existing Work

Traditional cricket analytics systems mainly rely on static statistics such as batting average, strike rate, economy rate, and player rankings. These systems do not account for the contextual factors that significantly influence a player's performance, such as the opponent's strength, venue characteristics, recent form, or match pressure.

Existing solutions also depend heavily on manual analysis by coaches and scouts, which is often subjective and inconsistent. Furthermore, most of the available software tools are proprietary and commercial, accessible only to elite professional teams.

CricNex addresses these limitations by introducing predictive modeling and automation into player performance evaluation. Unlike traditional systems, it uses machine learning regression and time-series forecasting models that learn from historical match data to predict future outcomes under specific conditions.

The proposed system thus represents a significant improvement over pre-existing work by providing:

- Automated, data-driven performance forecasting

- Context-aware evaluation of players
- A publicly accessible and research-oriented analytics tool

3.2.2 Perspective on Product

The CricNex system is a standalone web application that integrates machine learning models with a data visualization interface. The product's architecture follows a modular, full-stack design composed of:

1. Frontend (React.js):

The user interface allows coaches, analysts, and selectors to visualize performance graphs, compare players, and view prediction results interactively.

2. Backend (Flask Framework):

The backend provides RESTful APIs that handle prediction requests, process input data, and communicate with the trained machine learning models.

3. Database (MongoDB):

The system employs MongoDB to store historical match records, player profiles, model predictions, and performance logs in a flexible document-based schema.

4. Machine Learning Models:

The core intelligence of the product lies in its ML modules, which include Random Forest, XGBoost, ARIMA, and LSTM models. Each model is trained using cleaned and engineered player data and evaluated based on performance metrics such as MAE, RMSE, and R^2.

The system is designed to operate as a web-accessible analytics platform that can be deployed locally or on a cloud server.

3.2.3 Product Attributes

CricNex exhibits several defining attributes that enhance its usability, efficiency, and reliability:

- **Accuracy:** Employs multiple ML algorithms to ensure high predictive performance
- **Scalability:** Can be extended with additional models or larger datasets without redesign
- **User Interactivity:** Provides dynamic dashboards, graphs, and comparative analytics
- **Automation:** Performs preprocessing, training, and evaluation with minimal user intervention
- **Maintainability:** Codebase organized into modular components for easy updates and debugging
- **Portability:** Compatible with major operating systems (Windows, Linux, macOS)

3.2.4 End User Characteristics

The target end users of the CricNex application include:

- **Coaches and Team Selectors:** For evaluating players' readiness and potential for upcoming matches
- **Cricket Analysts and Statisticians:** For generating player insights and performance comparisons
- **Sports Enthusiasts and Researchers:** For exploring the predictive potential of machine learning in sports
- **Students and Academicians:** For learning about applied data science and model integration

The end users are expected to have a basic understanding of cricket performance metrics and general computer literacy. The system design ensures simplicity so that even non-technical users can access and interpret the predictive outcomes.

3.2.5 Operating Environment

CricNex runs in a web-based environment and supports both local and cloud deployments. The key components of the operating environment are:

- **Frontend:** React.js (JavaScript, HTML5, CSS3)
- **Backend:** Flask (Python)
- **Database:** MongoDB (NoSQL document database)
- **ML Frameworks:** Scikit-learn, XGBoost, TensorFlow, and Pandas for data manipulation and modeling
- **Deployment:** Can be hosted on local servers or cloud platforms such as AWS, Heroku, or Render
- **Browser Compatibility:** Supports Chrome, Edge, and Firefox browsers

3.2.6 Constraints in Design and Implementation

Several design and implementation constraints exist within the scope of this project:

- **Data Limitations:** Availability of complete and high-quality cricket datasets is limited
- **Computational Resources:** Model training and evaluation may require significant processing power and memory
- **Dependency Management:** Compatibility between ML libraries and Python versions must be maintained
- **Performance:** Time-series models like LSTM and ARIMA may have longer training durations

- **Scope Restriction:** The system focuses only on player-level predictions, not full match or team outcomes

3.2.7 Assumptions and Dependencies

The following assumptions and dependencies apply to the CricNex system:

Assumptions:

- The Kaggle dataset used contains accurate and up-to-date cricket player records
- The system assumes availability of a stable internet connection during online deployment
- The end user will have access to a modern web browser for using the application
- Users have basic understanding of cricket terminology and statistics

Dependencies:

- Python 3.9+ and Node.js installed properly in the development environment
- MongoDB database server accessible and properly configured
- External libraries such as Scikit-learn, TensorFlow, XGBoost, and Flask are available and compatible with the development system
- Kaggle datasets remain accessible for data collection

3.3 PRODUCT FUNCTIONALITY

3.3.1 Admin Module

The Admin Module provides complete control over the management and maintenance of the system. It is designed for administrative users who oversee the technical and operational aspects of the application. The main functions of this module include the following:

1. Dataset Management:

The administrator can upload, update, and validate datasets containing player statistics and match details. The module also allows cleaning and verification of incoming data before it is used for model training.

2. Model Training and Evaluation:

The admin can initiate training of machine learning models such as Random Forest, XGBoost, ARIMA, and LSTM. The system automatically performs preprocessing, feature scaling, and validation. Evaluation metrics including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2 are displayed for comparison.

3. User Access Management:

The administrator controls user permissions, ensuring that only authorized users can access prediction and comparison features.

4. System Monitoring:

The admin can monitor the system's performance, storage usage, and backend processes. Logs of predictions, data updates, and training results are maintained for record keeping.

5. Model Selection:

Based on performance metrics, the admin can choose the best-performing model for deployment to the prediction module.

Overall, the Admin Module ensures the reliability and consistency of the system by maintaining clean data, up-to-date models, and regulated access.

3.3.2 User Module

The User Module serves as the interface for analysts, coaches, and selectors to interact with the system. It enables them to perform player-level analysis and obtain predictive insights without requiring technical knowledge of the underlying machine learning models. The primary functions of this module are as follows:

1. Player Search and Input:

Users can search for a specific player or select one from a predefined list. Input parameters such as opponent, venue, and match conditions can be entered to generate performance predictions.

2. Prediction Generation:

The module interacts with the Flask backend to retrieve predictions for batting and bowling metrics including runs scored, wickets taken, and strike rate. The results are displayed in real time.

3. Comparative Analysis:

Users can compare the predicted performance of multiple players under similar or different conditions. This helps in team selection and tactical planning.

4. Visualization of Results:

The module provides graphical representations of trends, such as performance over time, consistency ratios, and venue-based averages. Interactive charts and graphs are rendered using Plotly and Recharts.

5. Performance Reports:

Users can view detailed reports summarizing player statistics, recent form indices, and opponent strength scores.

6. Model Interpretation:

To promote transparency, the system also displays feature importance rankings, allowing users to understand which factors most influence the predictions.

The User Module thus provides an intuitive and interactive interface for deriving actionable insights from complex machine learning outputs. It bridges the gap between technical model results and practical decision-making in cricket analytics.

3.4 EXTERNAL INTERFACE REQUIREMENTS

3.4.1 User Interfaces

The system provides a web-based interface developed using React.js for the frontend. The design focuses on usability, clarity, and simplicity to ensure that both technical and non-technical users can operate it effectively. The major elements of the user interface include:

- **Dashboard:** Displays a summary of players, recent predictions, and performance insights
- **Player Analysis Page:** Allows users to search for a player, view historical performance, and generate predictions
- **Visualization Area:** Presents performance graphs, comparisons, and trends using interactive charts created with Plotly and Recharts
- **Input Forms:** Enables users to select parameters such as opponent, venue, and recent match data for customized predictions
- **Admin Panel:** Restricted access section that allows the administrator to manage datasets, monitor models, and perform maintenance activities

The interface is responsive and optimized for web browsers, providing accessibility on both desktop and laptop devices.

3.4.2 Hardware Interfaces

CricNex is designed to run on standard computing devices with no need for special hardware interfaces. The recommended configuration for development and deployment is as follows:

- **Processor:** Intel Core i5 or higher
- **RAM:** Minimum 8 GB
- **Storage:** At least 256 GB of available space
- **Display:** Minimum resolution of 1366 × 768 pixels

Optional configurations include cloud-based environments such as AWS or Google Cloud, which may be used for hosting and scalability.

3.4.3 Software Interfaces

The system integrates several software components and frameworks for its operation. The major interfaces include:

- **Flask Framework:** Acts as the backend server and handles API requests from the frontend

- **MongoDB:** Serves as the primary database for storing player data, match records, and prediction results
- **React.js:** Manages the user interface, rendering all components dynamically through REST API calls
- **Python Libraries:** Utilizes Scikit-learn, XGBoost, TensorFlow, and Pandas for machine learning model development, data preprocessing, and feature engineering
- **Plotly and Recharts:** Used for visualization and representation of predictive outcomes in graphical form

Integration between these components is achieved through RESTful APIs, which facilitate communication between the frontend, backend, and machine learning modules.

3.4.4 Communication Interfaces

The system follows standard web communication protocols to ensure smooth and secure data exchange between different components. The details are as follows:

- **Protocol:** HTTP and HTTPS are used for communication between client and server
- **Data Format:** All data is transmitted in JSON format for consistency and compatibility
- **API Requests:** The frontend communicates with the Flask backend through RESTful API endpoints for retrieving predictions, historical data, and visual analytics
- **Database Connectivity:** The backend connects to the MongoDB database using the PyMongo library to read and write data efficiently
- **Network Requirements:** A stable internet connection is required for remote database access and online deployment

The communication interfaces are designed to be secure, lightweight, and extensible to support future enhancements such as mobile applications or additional analytical modules.

3.5 OTHER NON-FUNCTIONAL REQUIREMENTS

3.5.1 Performance Requirements

The performance of the CricNex system is critical to ensure accuracy, responsiveness, and scalability. The system is designed to handle moderate to large volumes of player and match data efficiently while maintaining acceptable response times. The key performance requirements are as follows:

- The system should be able to generate predictions within 2 seconds after input submission once the model is loaded
- The machine learning models should achieve an accuracy level of at least 80% on test datasets based on evaluation metrics such as MAE, RMSE, and R^2 score
- The database should handle queries for up to 10,000 player records without degradation in response time

- The web interface should remain responsive under concurrent user access
- Data retrieval and visualization operations must execute efficiently, ensuring a smooth user experience even when large datasets are used

These performance requirements ensure that the system remains reliable and effective for real-world analytical use.

3.5.2 Safety Requirements

The safety requirements ensure that user data, prediction results, and model parameters are stored and accessed securely without risk of corruption or unauthorized modification. The following measures are defined:

- Data stored in MongoDB must be regularly backed up to prevent loss due to system failures
- Administrative privileges must be restricted to authorized personnel to avoid accidental modification of datasets and models
- All communications between the frontend and backend should use secure HTTP (HTTPS) to prevent interception
- Access to model files and API endpoints must be protected through authentication mechanisms
- Error-handling procedures must ensure that unexpected failures do not lead to data corruption or application crashes

These safety protocols help in maintaining the stability and integrity of the system.

3.5.3 Software Quality Attributes

3.5.3.1 Reliability

The system must provide consistent and accurate predictions across different datasets and usage conditions. It should recover gracefully from unexpected interruptions and handle invalid user inputs effectively.

3.5.3.2 Usability

The user interface should be simple and intuitive, enabling non-technical users such as coaches and analysts to use the system without extensive training. Labels, navigation menus, and visualization components should be clearly designed for clarity and ease of use.

3.5.3.3 Efficiency

Machine learning algorithms and data-handling modules must be optimized to reduce computational load and minimize prediction time. Data preprocessing and feature engineering should be automated to enhance workflow efficiency. The overall system design should ensure minimal latency between data input, model execution, and output visualization.

3.5.3.4 Scalability

The architecture must support future scalability, allowing for the inclusion of additional datasets, players, and machine learning models. The modular design enables independent scaling of frontend, backend, and database components.

3.5.3.5 Maintainability

Code should follow modular programming practices to allow easy debugging, updates, and integration of new features. Clear documentation of APIs and model configurations must be maintained.

3.5.3.6 Security

User credentials, datasets, and prediction outputs must be stored securely. The system should implement access control mechanisms to prevent unauthorized data access. Database queries and input fields should be validated to prevent injection attacks.

3.5.3.7 Portability

The system must be compatible with all major operating systems such as Windows, Linux, and macOS. Browser compatibility should include Google Chrome, Microsoft Edge, and Mozilla Firefox.

3.5.3.8 Interoperability

CricNex must communicate effectively with other systems and data sources using standardized RESTful APIs and JSON data formats, making integration with third-party analytical tools feasible.

3.5.3.9 Performance

The system should maintain optimal response times and resource utilization across various operational scenarios, ensuring efficient execution of all computational tasks.

3.5.3.10 Flexibility

The design should allow future enhancements, such as integration of new algorithms, visualization tools, or modules for other sports analytics, without extensive code modifications.

3.6 SPECIFIC REQUIREMENTS

3.6.1 Operating Environment

3.6.1.1 Hardware

The CricNex system does not require specialized hardware and can run efficiently on a standard computing environment. The recommended hardware configuration for development and deployment is as follows:

- **Processor:** Intel Core i5 or higher
- **RAM:** Minimum 8 GB (16 GB recommended for model training)

- **Storage:** Minimum 256 GB of free disk space
- **Display:** 1366 × 768 resolution or higher
- **Network:** Stable internet connection for cloud deployment and data access

For large-scale training or high-performance tasks, cloud computing platforms such as AWS, Google Cloud, or Microsoft Azure may be used to provide additional computational resources.

3.6.1.2 Software

The software environment is a combination of open-source tools, libraries, and frameworks that support machine learning model development, database management, and frontend-backend integration. The following components are required:

- **Operating System:** Windows, Linux, or macOS
- **Programming Languages:** Python 3.9+ and JavaScript (ES6 or later)
- **Backend Framework:** Flask for API development and model integration
- **Frontend Framework:** React.js for user interface and visualization
- **Database:** MongoDB for data storage and retrieval
- **Development Tools:** Visual Studio Code, Git, and Postman for version control and API testing
- **Machine Learning Libraries:** Scikit-learn, XGBoost, TensorFlow, Pandas, NumPy
- **Visualization Libraries:** Plotly, Recharts, Matplotlib for performance visualization
- **Web Servers:** Gunicorn or Nginx for production deployment (optional)

All dependencies are managed through virtual environments to ensure compatibility and maintainability. The software stack ensures flexibility for both local and cloud-based deployment.

The system must be tested across multiple operating systems to verify cross-platform performance and browser compatibility. It should run efficiently on commonly used web browsers such as Google Chrome, Microsoft Edge, and Mozilla Firefox.

3.7 DELIVERY PLAN

Week	Task / Phase	Key Activities
Week 1	Requirement Analysis & Design	Define requirements, analyze feasibility, and design system architecture
Week 2	Data Collection & Preprocessing	Collect datasets, clean data, handle missing values, and prepare training sets

Week 3	Feature Engineering & Baseline Models	Create derived features and build baseline models using Random Forest and Linear Regression
Week 4	Advanced Model Implementation	Implement XGBoost, ARIMA, and LSTM models and perform performance evaluation
Week 5	Backend Development	Develop Flask APIs, integrate MongoDB, and manage model endpoints
Week 6	Frontend Development	Build React.js interface with dashboards and performance visualization
Week 7	Integration	Connect frontend and backend modules, test API functionality
Week 8	Testing & Optimization	Conduct testing, improve accuracy and response time
Week 9	Deployment & Documentation	Deploy on cloud platform and prepare final documentation