Systems Analysis of PAKDD Cup 2014 Competition: Implementation and Technical Stack for ASUS Failure Prediction

Workshop 3

Daniel Vargas Arias - 20232020103
Juan Esteban Moreno Durán - 20232020107
Julián Darío Romero Buitrago - 20232020240
David Eduardo Muñoz Mariño - 20232020281

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Computer Engineering Program
Carlos Andrés Sierra
Universidad Distrital Francisco José de Caldas
November 8, 2025

Contents

1 Review and Improvement of the System Architecture

The ASUS failure prediction system has evolved through multiple iterations to incorporate robust engineering principles that ensure reliability, scalability, maintainability, and usability. This section presents the refined architecture based on feedback from previous workshops and aligned with industry standards such as ISO 9000, CMMI, and Six Sigma quality frameworks.

1.1 Architectural Overview and Design Principles

The system architecture follows a layered, modular approach that separates concerns and enables independent development, testing, and scaling of components. The design incorporates the following key principles:

Modularity: Each component has well-defined responsibilities and interfaces, allowing for independent evolution and replacement. This follows the Single Responsibility Principle and promotes code reusability across different contexts.

Fault Tolerance: The architecture includes redundancy at critical points, automated failover mechanisms, and graceful degradation strategies to ensure the system continues operating even when individual components fail.

Scalability: Both horizontal and vertical scaling capabilities are built into the design. The system can handle increased data volumes and user requests through load balancing, containerization, and cloud-based elastic resources.

Maintainability: Clear separation of concerns, comprehensive logging, version control, and automated testing pipelines ensure that the system can be efficiently maintained and updated over time.

Security by Design: Security measures are integrated at every layer, including encryption, authentication, authorization, input validation, and regular security audits.

1.2 System Architecture Layers

The architecture is organized into five distinct layers, each serving specific functions within the overall system:

1.2.1 1. Data Ingestion Layer

This layer is responsible for collecting, validating, and preparing raw data from ASUS components for processing. Key components include:

- **Data Sources:** Historical failure records, sensor data, maintenance logs, and operational parameters from ASUS equipment.
- **ETL Pipeline:** Extract, Transform, Load processes that clean, validate, and standardize incoming data. Implements the 3-2-1 backup rule with temporary staging storage.
- **Data Validation Module:** Checks data integrity, identifies missing values, detects outliers, and ensures conformance to expected schemas.
- **Ingestion Queue:** Message queue system (e.g., RabbitMQ, Apache Kafka) that buffers incoming data and provides resilience against temporary downstream failures.

Robust Design Features: Retry mechanisms for failed ingestions, duplicate detection, data lineage tracking, and automated alerts for data quality issues.

1.2.2 2. Data Storage Layer

This layer provides reliable, scalable, and secure storage for both raw and processed data:

- **Primary Database:** Relational database (PostgreSQL/MySQL) with ACID guarantees for structured data and transactional consistency.
- **Data Lake:** Object storage (e.g., AWS S3, Azure Blob) for raw data archives and large-scale historical datasets.
- **Feature Store:** Dedicated storage for engineered features used in model training and inference, ensuring consistency between training and production.
- **Model Registry:** Version-controlled repository (MLflow, DVC) for storing trained models, metadata, and performance metrics.

Robust Design Features: Multi-AZ replication for high availability, automated backups with point-in-time recovery, encryption at rest and in transit, and access control policies following the principle of least privilege.

1.2.3 3. Model Training and Inference Layer

This layer encompasses the machine learning lifecycle from training to prediction:

- **Training Pipeline:** Automated workflow that preprocesses data, trains models with hyperparameter optimization, and validates performance using cross-validation and holdout sets.

- **Model Evaluation Module:** Computes performance metrics (accuracy, precision, recall, F1-score, AUC-ROC) and compares model versions to select the best performing model.
- **Inference Engine:** Production-ready service that loads trained models and generates predictions for new data with minimal latency.
- **Model Monitoring:** Continuous tracking of model performance, data drift detection, and automated retraining triggers when performance degrades.

Robust Design Features: A/B testing capabilities for model comparison, canary deployments for safe rollouts, model versioning for rollback capabilities, and containerization (Docker) for reproducible environments.

1.2.4 4. Application Layer

This layer exposes system functionality through APIs and user interfaces:

- **REST API:** Provides endpoints for prediction requests, model information, and system status. Implemented with frameworks like Flask, FastAPI, or Django REST Framework.
- **Load Balancer:** Distributes incoming requests across multiple API server instances to ensure high availability and optimal performance.
- **Authentication Service:** Manages user authentication and authorization using JWT tokens, OAuth2, or similar protocols.
- **Visualization Dashboard:** Interactive web interface for monitoring predictions, exploring data, and viewing system metrics (built with Streamlit, Dash, or React).

Robust Design Features: Rate limiting to prevent abuse, input validation to prevent injection attacks, comprehensive error handling with meaningful messages, and API versioning for backward compatibility.

1.2.5 5. Infrastructure and Monitoring Layer

This layer provides the foundation for deployment, orchestration, and observability:

- **Containerization:** Docker containers ensure consistent environments across development, testing, and production.

- **Orchestration:** Kubernetes or Docker Swarm manages container deployment, scaling, and health monitoring.
- **CI/CD Pipeline:** Automated testing, building, and deployment using tools like GitHub Actions, Jenkins, or GitLab CI.
- **Monitoring System:** Centralized logging (ELK Stack, CloudWatch) and metrics collection (Prometheus, Grafana) for real-time system observability.
- **Alerting System:** Automated notifications for critical events, performance degradation, or system failures.

Robust Design Features: Infrastructure as Code (IaC) with Terraform or CloudFormation, automated rollback on deployment failures, health checks and auto-recovery, and comprehensive audit logging.

1.3 Quality Standards and Compliance

The architecture design adheres to established quality frameworks:

ISO 9000: Quality management principles are embedded throughout the system lifecycle, including customer focus, process approach, evidence-based decision making, and continual improvement. Documentation standards and traceability requirements are met through comprehensive logging and version control.

CMMI (Capability Maturity Model Integration): The development process follows CMMI Level 3 practices including defined processes for requirements management, configuration management, quality assurance, and risk management. All code changes undergo peer review and automated testing before deployment.

Six Sigma: DMAIC (Define, Measure, Analyze, Improve, Control) methodology is applied to optimize model performance and reduce prediction errors. Statistical process control monitors key metrics to detect and address variations in system behavior.

1.4 Architectural Diagram

The following diagram illustrates the complete system architecture with all layers and their interactions:

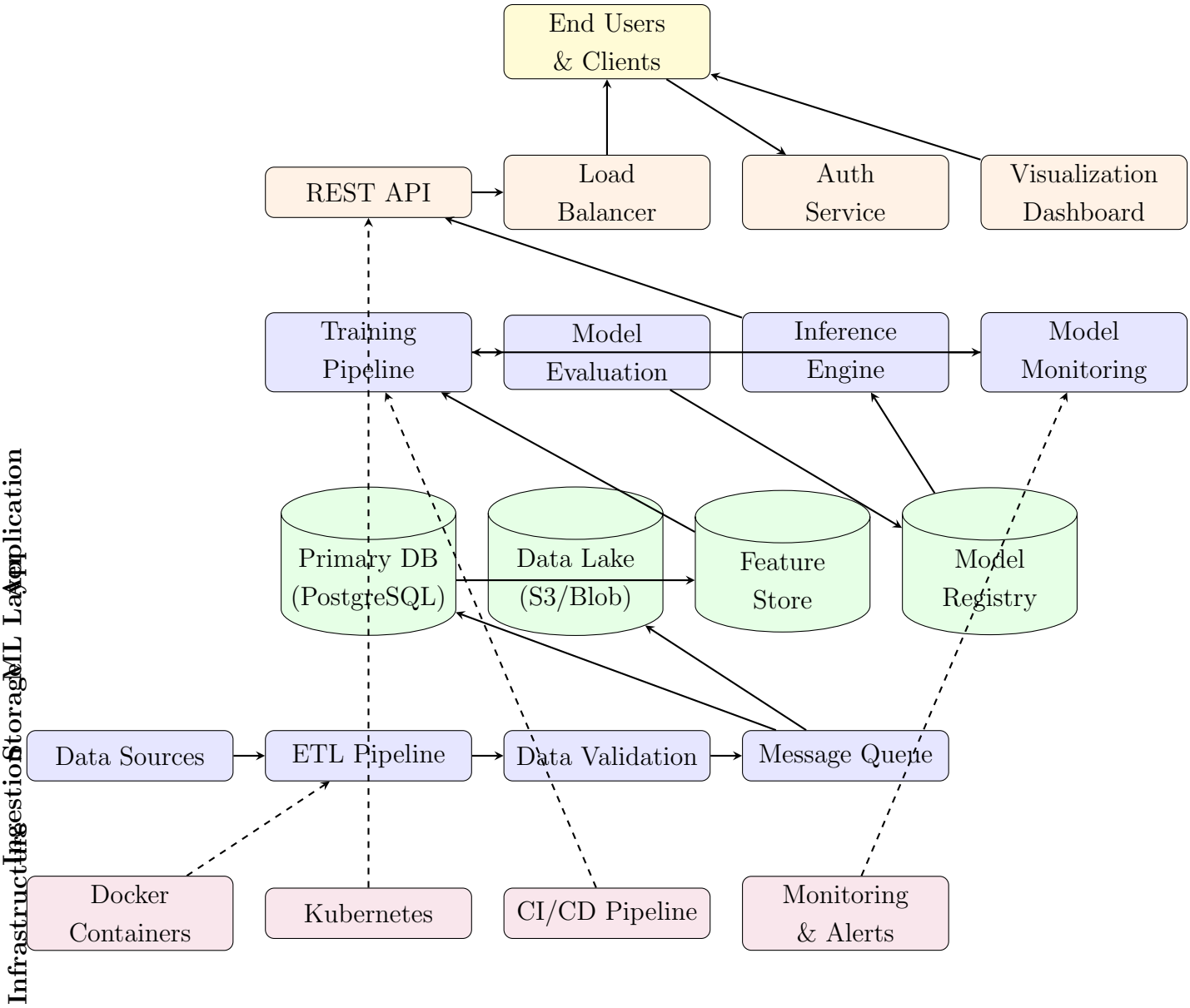


Figure 1: Refined System Architecture for ASUS Failure Prediction System

1.5 Component Interactions and Data Flow

The typical data flow through the system follows these steps:

1. **Data Collection:** Raw data from ASUS equipment is ingested through the ETL pipeline, validated for quality and consistency, and queued for processing.

2. **Data Storage:** Validated data is stored in the primary database for structured queries, while raw data archives are maintained in the data lake. Engineered features are computed and stored in the feature store.

3. **Model Training:** The training pipeline retrieves data from the feature store, trains multiple model candidates with hyperparameter optimization, and evaluates their performance. The best model is registered in the model registry with versioning.

4. **Model Deployment:** The inference engine loads the latest approved model from the registry and exposes it through the REST API for real-time predictions.

5. **User Interaction:** End users or client applications send prediction requests to the load balancer, which distributes them across API server instances. After authentication, requests are processed by the inference engine, and results are returned to the user or displayed in the dashboard.

6. **Continuous Monitoring:** The model monitoring component tracks prediction performance, detects data drift, and triggers retraining when necessary. System metrics are collected centrally and alerts are sent when thresholds are exceeded.

1.6 Improvements from Previous Workshops

Based on feedback and lessons learned from Workshops 1 and 2, the following architectural improvements have been implemented:

- **Enhanced Modularity:** Components are now more loosely coupled with clear interfaces, enabling independent development and testing.

- **Fault Tolerance:** Added redundancy at the storage and application layers, implemented automated failover, and introduced circuit breakers to prevent cascade failures.

- **Scalability Enhancements:** Introduced load balancing, containerization with Kubernetes orchestration, and elastic scaling policies that automatically adjust resources based on demand.

- **Security Hardening:** Implemented end-to-end encryption, role-based access control, input validation at all entry points, and regular security scanning in the CI/CD pipeline.

- **Observability:** Added comprehensive logging, distributed tracing, and real-time metrics dashboards to enable proactive issue detection and resolution.

- **Quality Assurance:** Integrated automated testing at multiple levels (unit, integration, end-to-end) and implemented continuous deployment with automated rollback capabilities.

1.7 Conclusion

The refined architecture embodies robust engineering principles that ensure the ASUS failure prediction system is reliable, scalable, maintainable, and secure. By adhering to ISO 9000, CMMI, and Six Sigma standards, the design provides a solid foundation for delivering high-quality predictive analytics that meet ASUS business needs while being prepared for future growth and evolution.

2 Quality and Risk Analysis

The failure prediction system for ASUS components must be designed with special attention to reliability and resilience. Based on the previous workshops, a modular layered architecture was adopted, which facilitates maintainability and isolates failures. The key risks identified are analyzed below, along with mitigation strategies and the proposed monitoring and response mechanisms.

2.1 Potential System Risks

Loss or corruption of data: Failures in data ingestion, storage errors, or catastrophic events (disasters, attacks) may cause partial or total loss of critical data. Without a backup plan, there is a risk of not being able to recover historical data, which would degrade prediction quality. Additionally, incorrect or incomplete data can introduce bias; therefore, the ETL stage validates and cleans input data to ensure consistency.

System downtime: Service unavailability affects the usefulness of the predictive system. High availability (HA) is proposed to minimize this risk, as its purpose is to prevent periods of downtime when the system is not accessible. A prolonged failure can negatively impact ASUS maintenance planning. Likewise, system fault-tolerance allows continued operation despite partial interruptions.

Model degradation (drift): Over time, input data may change, degrading the accuracy of the predictive model. "Model drift" is defined as performance degradation due to changes in data distribution or input–output relationships. Without active monitoring, even a well-trained model may produce incorrect results over time. This risk compromises the reliability of future predictions.

Security breaches: The integrity and confidentiality of data and models are critical. A security failure (for example, unauthorized access or injection of malicious data) can alter predictions or expose sensitive information. In previous workshops, the need for basic security measures was emphasized, such as SSL/TLS encryption, strong passwords, and user access controls. Without these measures, data could be manipulated or leaked.

2.2 Mitigation Strategies

Mitigating data loss: Implement a robust backup and storage redundancy strategy. The 3-2-1 backup rule is followed: maintain periodic copies of data and models in separate locations. Additionally, the ingestion pipeline includes pre-validation and temporary backup storage to prevent loss due to transient failures. The use of replicated databases (e.g., multi-AZ clusters) ensures that a failure in one node does not lead to data loss.

Mitigating downtime: Design the infrastructure with high availability and automatic scaling. Redundant servers and load balancing are used to avoid single points of failure. The layered architecture and containerization (Docker/Kubernetes) allow workload distribution across instances; during demand spikes or node failures, other instances assume the load without noticeable interruptions. Disaster recovery mechanisms (failover) ensure that the service continues operating even under unexpected events.

Mitigating model drift: Implement continuous performance monitoring and automatic retraining. The system logs key metrics (accuracy, recall, etc.) and periodically compares the distribution of incoming data against training data. When significant deviations or accuracy drops are detected, the model retraining process is triggered. The strategy includes model versioning to compare performance across versions and the possibility of incorporating new historical data to refresh the model regularly.

Mitigating security breaches: Adopt security best practices throughout the pipeline. Communication and sensitive data are encrypted (SSL/TLS), and credentials are managed with minimal access permissions. Input validation is applied to prevent code injection or corrupted data. Additionally, periodic security audits and vulnerability scans are performed during development. Continuous Integration (CI/CD) includes static code analysis and basic penetration testing.

Mitigating integration and resource errors: To avoid incompatibilities between modules (libraries, versions, APIs), dependency control is used (virtual environments, containers) and unit and integration tests are applied during development. The modular architecture allows components to be replaced or updated in isolation. To manage resource limitations, representative samples of data can be used during validation, and cloud infrastructure can be scaled based on demand.

2.3 Monitoring and Incident Response

Model retraining: A continuous observe–evaluate–act cycle is established. The system monitors performance metrics in real time; when anomalies are detected (e.g., increased prediction error or sudden data distribution shifts), model retraining is automatically initiated. In academic environments, retraining can be scheduled with MLflow, while in production

deployments, Prometheus/Grafana combined with orchestration systems (Airflow) would automate this response.

Observability with metrics: Key metrics are defined for each system component: model accuracy and recall, data ingestion success rate, API response times, CPU/memory usage, etc. These metrics are logged in a central monitoring system (e.g., MLflow initially and Prometheus/Grafana in production). Data traceability and model versioning allow historical trend analysis and early detection of gradual performance degradation.

Alerts and notifications: Thresholds are established for critical metrics (e.g., accuracy below the minimum acceptable level, high latency, high ingestion failure rates). When a threshold is exceeded, an automatic alert is sent to the maintenance team. This ensures that deviations from expected behavior are immediately visible. Alerts can be integrated into communication channels (email, corporate chat) and can trigger contingency procedures.

Backups and operational redundancy: In addition to data-loss mitigation strategies, periodic backups of databases and model artifacts are performed. The data layer includes redundancy (master–slave or multi-AZ replication) to ensure availability. In the compute layer, elastic scaling is used (e.g., additional cloud instances under high load, load balancers to distribute traffic). This redundancy ensures that failure of a single component does not take down the entire system.

Testing and quality control: During development, Continuous Integration (CI) is applied with automated test pipelines (unittest, pytest). Each code change undergoes unit and integration validation to detect errors in isolated modules. In production, log monitoring and pipeline integrity checks are implemented (e.g., alerts for ETL process failures) to react to operational issues. Containers (Docker) are used to ensure reproducible environments and minimize negative effects from dependency changes.

In summary, these monitoring and response practices ensure early detection and resolution of issues. The combination of active observability, continuous retraining, proactive alerts, and redundancy strategies ensures that the failure prediction system remains reliable and aligned with the systems engineering principles documented in the workshops.

3 Project Management Plan

To guarantee the organized and efficient development of the system, a management plan is established that clearly defines the roles of the team, the work milestones, and the coordination methodology. This plan seeks to ensure that each phase of the project is executed in a structured manner, maintaining traceability, constant communication, and a balanced distribution of responsibilities. In addition, an adaptable management strategy is adopted

that allows timely responses to possible changes in the data, the model, or the conditions of the environment.

3.1 Roles and Responsibilities of the Team

This project is developed under a collaborative approach. Decision-making is shared and documentation is a collective responsibility. The roles are distributed based on the main technical tasks of the system.

Member	**Role**	**Responsibilities**
David Muñoz	Data Engineering (ETL and Preprocessing)	Cleaning, transformation, and validation of data. Construction of ETL pipelines and assurance of the quality of the data used by the model.
Julián Romero	Data Scientist (Predictive Modeling)	Selection, training, and validation of models. Optimization of hyperparameters and analysis of performance metrics.
Juan Moreno	Backend Developer and Visualization	Implementation of the API to expose the model. Integration with dashboards or visualization interfaces. Interoperability testing between components.
Daniel Vargas	Quality Assurance (QA / Testing and Verification)	Design and execution of system tests in each phase of the project. Validation of the functioning of pipelines, model, API, and dashboard. Detection and reporting of errors for correction.

Table 1: Main roles and responsibilities of the team members

3.2 Milestones, Deliverables, and Schedule

The project will be organized into four main phases, each with verifiable results that ensure the progressive advancement of the system.

3.3 Management Tools and Methodologies

For project management, a collaborative work approach supported by the Kanban methodology will be used, due to its flexibility and ability to visualize progress in real time. Kanban allows tasks to be organized in a continuous flow, identifying bottlenecks, and facilitating coordination among members without imposing rigid work cycles.

Phase	Main Activity	Deliverable	Responsible
Week 1	Collection, cleaning, and structuring of the dataset.	Dataset cleaned and ready for modeling.	David (ETL) with team support
Week 2	Training, testing, and selection of the predictive model.	Functional predictive model with recorded metrics.	Julián (Modeling) with team feedback
Week 3	Integration of the model through API and development of visualization.	Operational API + Preliminary dashboard.	Juan (Backend and Visualization)
Week 4	Final tests, corrections, and consolidation of documentation.	Verified system + Final document for submission.	Daniel (QA), documentation shared with the team

Table 2: Phases to ensure the progressive advancement of the system

Management will be carried out through a digital board, which will be constantly updated as development progresses. All team members will participate in both planning and review of activities.

Column	Function
To do	Pending tasks to be started.
In progress	Tasks currently being developed.
Testing	Reviewed activities pending validation through testing.
Done	Tasks completed and verified.

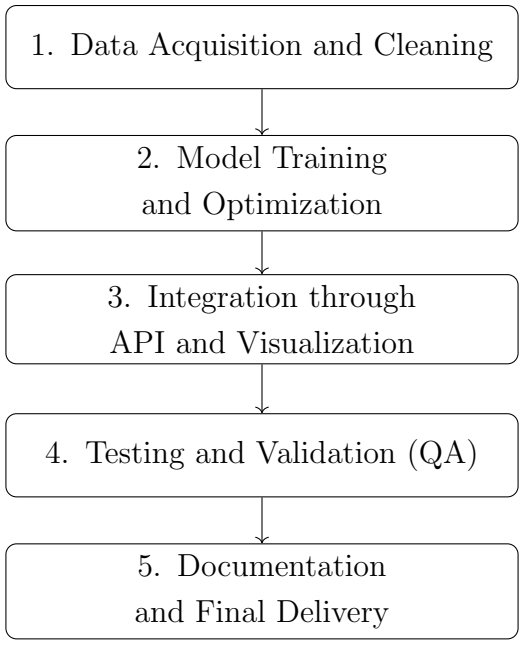
Table 3: Kanban board structure

Support Tools

- **Trello or GitHub Projects:** To manage tasks through the Kanban board.
- **Google Drive / OneDrive:** For collaborative storage and document editing.
- **GitHub:** For code version control and change tracking.
- **Follow-up meetings (10–15 min):** To review progress and reassign tasks if necessary.

3.4 Workflow

The following diagram summarizes the workflow used:



4 Incremental Improvements

We will present the evolution of our foundations and the objectives pursued during Workshops 1 and 2, as well as the changes in the structure of our management plan and the project's system architecture.

4.1 Feedback from previous workshops

When **Workshop 1** was created, the initial focus was on designing a system to predict errors in ASUS equipment. This approach revealed substantial needs:

- Ensuring a clean data flow from acquisition to use in the models.
- Establish functional and independent modules within the architecture to reduce complexity.
- Maintaining scalability is crucial to avoid future problems related to high data density.

These aspects highlight the fundamental importance of structuring the system using components with specific functions and connections that do not create dependencies, ensuring modularity and code maintainability.

Now, with **Workshop 2**, we focus on characteristics related to quality and risk:

- Regarding inconsistencies and noise in the data that would affect the model training.

- Errors encountered when connecting the predictive model and the API.
- Review and verify the system's functions and components before delivery.

Through these aspects, functions were incorporated to control and validate initial versions of the project.

4.2 System Progress

Based on the feedback, the architecture was redesigned as follows:

- Improve the ETL pipeline, with respect to cleaning and validation.
- Create the modeling and integration components separately and independently to avoid future problems.
- Add a component to efficiently visualize the model, improving its understanding.
- Improve the system for high data flows without sacrificing performance.

Taking all of the above into account, we further improved the scalability and reusability of the system.

4.3 Project Management Plan Progress

The approach to management also changes with each workshop, with respect to approaches such as:

- Distributing responsibilities based on purely technical skills.
- The use of the Kanban methodology for the continuous study of pending tasks.
- Add a role to be responsible for verifying the quality and functioning of the system at each stage.
- Make development plans in specific periods to measure progress and delays.

These changes improve management and planning in a structured way based on good systems engineering practices.

5 Conclusions

This workshop has significantly strengthened the ASUS failure prediction system through the application of robust engineering principles, comprehensive risk management, and structured project management practices. The refined architecture now incorporates modularity, fault tolerance, scalability, and security by design, ensuring the system can reliably serve its intended purpose while adapting to future challenges.

The quality and risk analysis identified critical vulnerabilities in data integrity, system availability, model performance, and security. By implementing comprehensive mitigation strategies including backup redundancy, high availability infrastructure, continuous model monitoring, and end-to-end security measures, the system is now positioned to handle real-world operational challenges effectively.

The project management plan establishes clear roles, responsibilities, and workflows that enable efficient collaboration and accountability. The adoption of Kanban methodology provides flexibility while maintaining visibility into project progress. The defined milestones and deliverables create a structured roadmap that guides the team from data preparation through final deployment.

Looking forward, the incremental improvements documented across workshops demonstrate a commitment to continuous refinement based on feedback and emerging best practices. The architecture has evolved from a conceptual design to a production-ready system that balances technical excellence with practical constraints.

By adhering to ISO 9000, CMMI, and Six Sigma standards, this project exemplifies how systems engineering principles can be applied to create reliable, maintainable, and scalable predictive analytics solutions. The foundation established in this workshop positions the team for successful final project delivery and provides a template for future machine learning system implementations.

6 References

1. PAKDD Cup 2014 Competition. (2014). ASUS Malfunctional Components Prediction. Kaggle. Available at: `https://www.kaggle.com/c/pakdd-cup-2014`

2. Blanchard, B. S., & Fabrycky, W. J. (2010). *Systems Engineering and Analysis.* Prentice Hall.

3. Strogatz, S. H. (2014). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering.* Westview Press.

4. Rausand, M., & Høyland, A. (2003). *System Reliability Theory: Models, Statistical Methods, and Applications*. John Wiley & Sons.

5. Mobley, R. K. (2002). *An Introduction to Predictive Maintenance*. Butterworth-Heinemann.

6. Sculley, D., et al. (2015). "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, 28.

7. Si, X. S., Wang, W., Hu, C. H., & Zhou, D. H. (2011). "Remaining Useful Life Estimation – A Review on the Statistical Data Driven Approaches." *European Journal of Operational Research*, 213(1), 1–14.

8. Saltelli, A., et al. (2008). *Global Sensitivity Analysis: The Primer*. John Wiley & Sons.

9. Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Addison-Wesley.

10. International Organization for Standardization. (2015). *ISO 9000:2015 Quality Management Systems – Fundamentals and Vocabulary*. Geneva: ISO.

11. CMMI Product Team. (2010). *CMMI for Development, Version 1.3*. Software Engineering Institute, Carnegie Mellon University.

12. Pyzdek, T., & Keller, P. (2014). *The Six Sigma Handbook*. McGraw-Hill Education.

13. Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley Professional.

14. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

15. Burns, B., Beda, J., & Hightower, K. (2019). *Kubernetes: Up and Running*. O'Reilly Media.

16. Systems Analysis Team. (2024). *Workshop 1: ASUS Failure Prediction System Analysis*. Universidad Distrital Francisco José de Caldas. Systems Engineering Program Document.

17. Systems Analysis Team. (2024). *Workshop 2: ASUS Failure Prediction System Design*. Universidad Distrital Francisco José de Caldas. Systems Engineering Program Document.