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

Workshop 2

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

Computer Engineering Program
Carlos Andrés Sierra
Universidad Distrital Francisco José de Caldas
September 27, 2025

Contents

1 Review of Workshop #1 Findings

1.1 Main System Analysis Findings

The analysis of the ASUS malfunctioning components prediction system reveals a complex socio-technical system combining historical data, predictive models, and business processes. The most notable findings include:

1.1.1 System Strengths

- Extensive historical database facilitating predictive modeling.
- Well-defined scope with measurable success indicators.
- Continuous feedback between detected failures and manufacturing improvements.
- High business value by optimizing production and inventory management.

1.1.2 Critical Weaknesses and Constraints

- High sensitivity to external factors such as temperature, humidity, or usage conditions.
- Complex interdependencies between components making precise prediction difficult.
- Need for substantial computational resources to process real-time data.
- Technical limitations due to sensor noise, incomplete data, and legacy system compatibility.
- Business constraints such as budget, regulatory compliance, and implementation timelines.

1.1.3 Data Characteristics

The dataset includes information on:

- Historical component failures, device age, and usage patterns.
- Environmental factors and manufacturing details.
- Input variables (specifications, production batches) and target variables (component failure or not).

The data exhibits high variability and nonlinear behavior, requiring models that handle uncertainty and hidden correlations between variables.

1.1.4 Chaos-Related Factors

The system exhibits chaotic behavior traits:

- Sensitive dependence on initial conditions: small variations in manufacturing or usage can produce entirely different failure trajectories.
- Positive and negative feedback loops: predictive models can both prevent failures and alter system behavior.
- Stochastic elements: randomness affects component wear, environment, and user behavior.
- Emergence of unforeseen global behaviors from local component interactions.

1.1.5 Design Proposals in Relation to Findings

To address these results, design ideas should focus on:

- Rigorous data quality management, with continuous validation and cleaning.
- Hybrid models combining deterministic and probabilistic approaches.
- Scalable architecture allowing real-time data processing.
- Adaptive systems with continuous learning and feedback from results.
- Controlled degradation mechanisms to maintain functionality during partial failures.
- Human supervision for critical decisions, mitigating risks of automated errors.

2 System Requirements Definition

Based on the conducted system analysis, the following measurable design requirements and user-centered needs were defined to ensure the ASUS component failure prediction system meets its objectives efficiently, reliably, and understandably.

2.1 Functional Requirements

1. **Accurate Failure Prediction:** The system must predict potential component failures with at least 85% accuracy within a specific time frame.

2. **Large-Scale Data Processing:** Must handle millions of historical records and real-time data streams from sensors and service reports.

3. **Automatic Model Updating:** The system should periodically retrain predictive models with new data, ensuring continuous performance improvement.

4. **Integration with Existing Systems:** Must interact with ERP systems, manufacturing databases, and service platforms without disrupting operations.

5. **Reporting and Alerts:** Must generate automatic failure risk notifications and visual reports for maintenance and inventory teams.

2.2 Non-Functional Requirements

1. **Performance:** System responses for predictions must occur within 1 second (sub-second for critical operations).

2. **Reliability:** 99.9% system availability required for real-time operations.

3. **Scalability:** Architecture must allow addition of new devices and data sources without affecting global performance.

4. **Model Accuracy and Stability:** Models must maintain error variance below 5% after each update or training cycle.

5. **Information Security:** Manufacturing, user, and failure data must be encrypted and protected per industrial and client privacy standards.

2.3 User-Centered Requirements

1. **Usability:** System interfaces (dashboards, reports, visualizations) must be intuitive and accessible for technical and administrative users.
2. **Model Interpretability:** Prediction results must include clear explanations (e.g., variables influencing failures) to aid decision-making.
3. **Decision Support:** System should provide automatic recommendations on preventive maintenance, repair prioritization, and component replacement.
4. **Adaptability:** System should automatically adjust to new manufacturing conditions or user behavior, reducing reliance on manual interventions.

2.4 Evaluation Metrics

- Model Accuracy $\geq$ 85%
- Response Time $\leq$ 1 second
- System Availability $\geq$ 99.9%
- Model Update $\leq$ 24 hours after receiving new data
- User Satisfaction $\geq$ 90% (per internal surveys)

3 High-Level Architecture

High-Level Architecture:

- Propose an architectural diagram showing the data flow and interaction between components.
- Label each module and briefly describe its responsibility (extraction, transformation, modeling, etc.).
- Mention how systems engineering principles influenced these structural decisions.

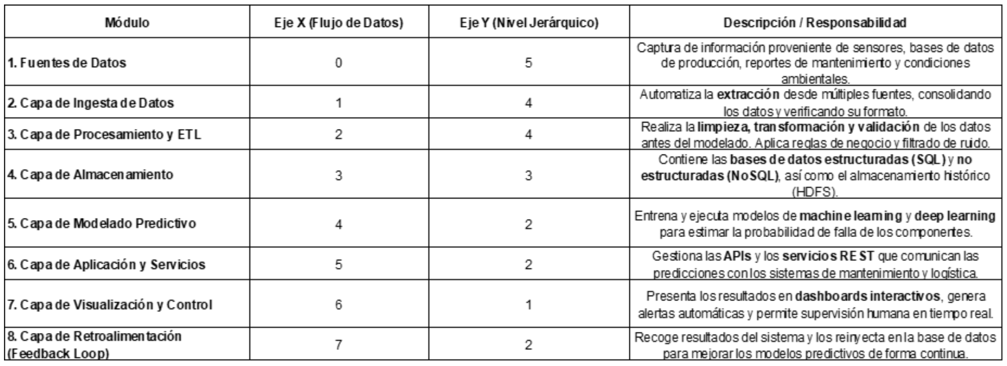
The high-level architecture of the ASUS component failure prediction system was designed based on systems engineering principles, applying a modular structure that facilitates traceability, scalability, and integration between the different subsystems.

The model consists of several functional layers that interact in an orderly manner to capture, process, store, and analyze data with the objective of anticipating failures in electronic components.

3.1 Component Representation and Hierarchy

Table 1 shows the functional location of each module within the system. Each module was assigned a pair of coordinates (X, Y), where:

- X represents the horizontal data flow (from acquisition to feedback).
- Y represents the functional hierarchical level of the system (from the highest to the lowest level of abstraction).



Módulo	Eje X (Flujo de Datos)	Eje Y (Nivel Jerárquico)	Descripción / Responsabilidad
1. Fuentes de Datos	0	5	Captura de información proveniente de sensores, bases de datos de producción, reportes de mantenimiento y condiciones ambientales.
2. Capa de Ingesta de Datos	1	4	Automatiza la **extracción** desde múltiples fuentes, consolidando los datos y verificando su formato.
3. Capa de Procesamiento y ETL	2	4	Realiza la **limpieza, transformación y validación** de los datos antes del modelado. Aplica reglas de negocio y filtrado de ruido.
4. Capa de Almacenamiento	3	3	Contiene las **bases de datos estructuradas (SQL)** y **no estructuradas (NoSQL)**, así como el almacenamiento histórico (HDFS).
5. Capa de Modelado Predictivo	4	2	Entrena y ejecuta modelos de **machine learning** y **deep learning** para estimar la probabilidad de falla de los componentes.
6. Capa de Aplicación y Servicios	5	2	Gestiona las **APIs** y los **servicios REST** que comunican las predicciones con los sistemas de mantenimiento y logística.
7. Capa de Visualización y Control	6	1	Presenta los resultados en **dashboards interactivos**, genera alertas automáticas y permite supervisión humana en tiempo real.
8. Capa de Retroalimentación (Feedback Loop)	7	2	Recoge resultados del sistema y los reinyecta en la base de datos para mejorar los modelos predictivos de forma continua.

Figure 1: X–Y Coordinates of the Modules of the Failure Prediction System

3.2 Bar Chart: Hierarchical Distribution

Figure 2 represents the system architecture using a bar chart, where each bar indicates the hierarchical position of a module according to its X–Y coordinates. The X-axis shows the sequence of the data flow (from acquisition to feedback), while the Y-axis expresses the functional level of the module within the architectural hierarchy.

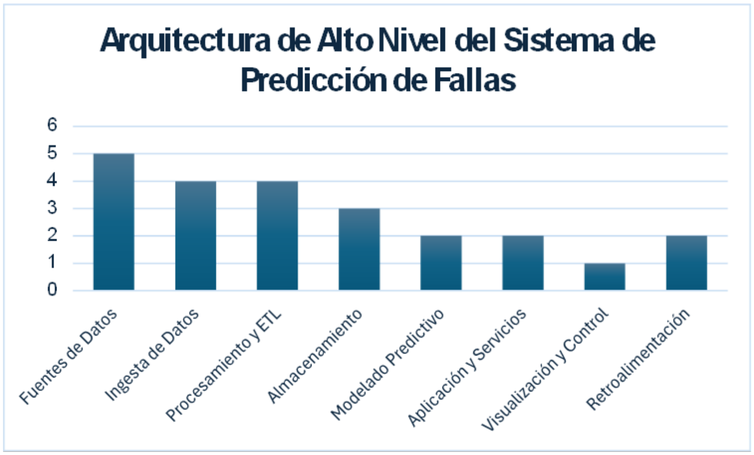


Figure 2: Bar Chart of the High-Level Architecture of the Failure Prediction System

The chart illustrates the hierarchical position of the system modules according to the X–Y coordinates established in Table 1. Data Sources occupy the highest level (Y=5), while Visualization and Control modules are at the lowest level (Y=1), showing the logical progression from capture to decision-making.

Data Flow Description:

1. **Extraction (X = 0 to 1):** Data is collected from industrial sensors and ERP systems. Variables include temperature, usage cycles, failure time, and environmental conditions.

2. **Transformation (X = 1 to 3):** The ETL layer cleans and structures the data. Inconsistencies are removed, outliers corrected, and normalized results are stored.

3. **Modeling and Prediction (X = 3 to 5):** Machine learning models process the cleaned data to predict the failure probability of each component.

4. **Application and Visualization (X = 5 to 7):** Results are exposed via web services and dashboards. The system adjusts its parameters through feedback.

3.3 Pie Chart: Functional Participation of Modules

The pie chart complements the architecture by showing the functional proportion of each module. Each section represents the relative weight of a layer in overall operation. Processing and ETL, Predictive Modeling, and Visualization and Control occupy a significant proportion.

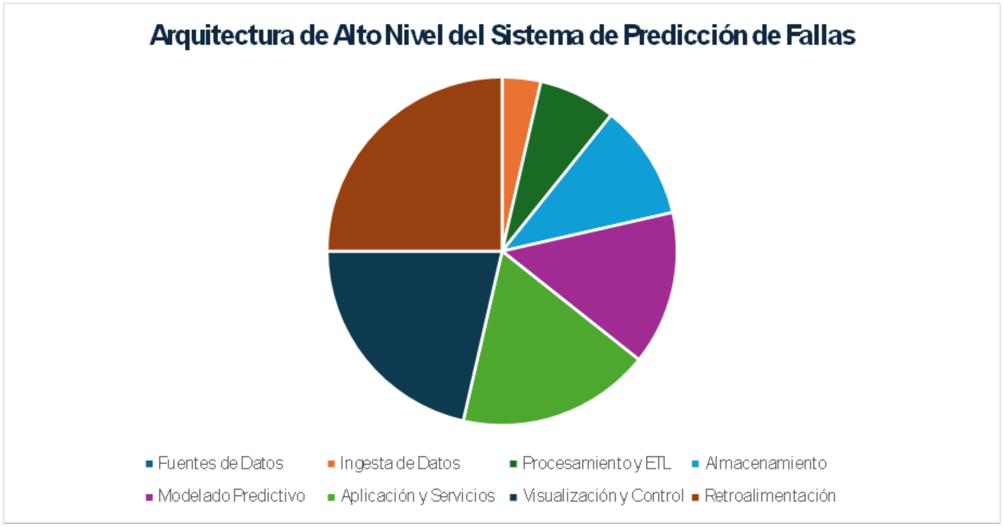


Figure 3: Pie Chart of the High-Level Architecture of the Failure Prediction System

3.4 Sequence Diagram: Interaction between Components

The sequence diagram (Figure 4) shows dynamic interaction between modules along the data processing flow. Participants include Data Sources, Data Ingestion, Processing and ETL, Storage, Predictive Modeling, Application and Services, Visualization and Control, and Feedback.

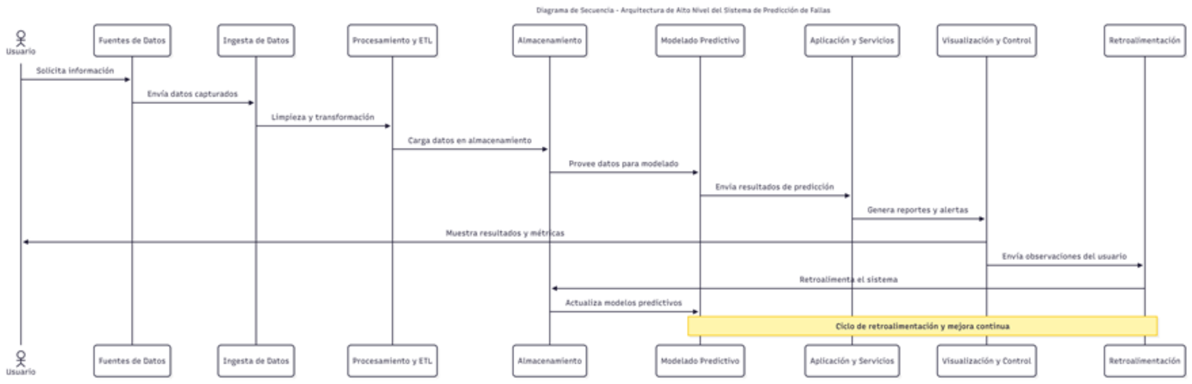


Diagrama de Secuencia - Arquitectura de Alto Nivel del Sistema de Predicción de Fallas
Usuario
Fuentes de Datos
Ingesta de Datos
Procesamiento y ETL
Almacenamiento
Modelado Predictivo
Aplicación y Servicios
Visualización y Control
Retroalimentación
Solicita información
Envía datos capturados
Limpieza y transformación
Carga datos en almacenamiento
Provee datos para modelado
Envía resultados de predicción
Genera reportes y alertas
Muestra resultados y métricas
Envía observaciones del usuario
Retroalimenta el sistema
Actualiza modelos predictivos
Ciclo de retroalimentación y mejora continua

Figure 4: Sequence Diagram of the Failure Prediction System

3.5 Flow Diagram of the Failure Prediction System

The flow diagram complements the architecture showing the logical path that data follows from capture to feedback. Each process represents a system layer and arrows indicate flow direction. This diagram is fundamental in systems engineering for visualizing process control, decisions, and module integration.

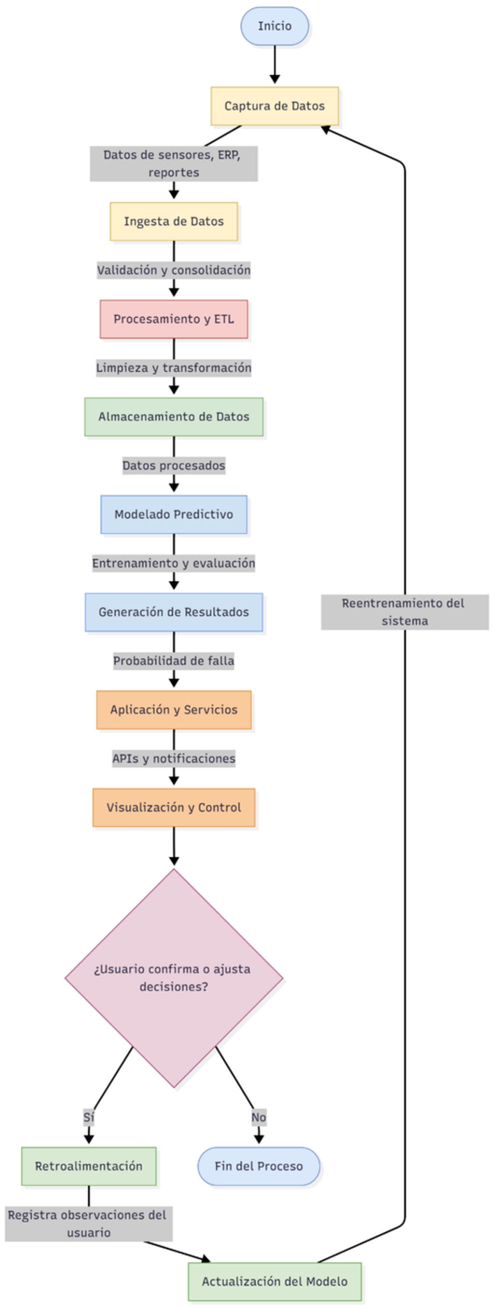


Figure 5: Flow Diagram of the Failure Prediction System

Flow Explanation:

1. **Start → Data Capture:** Acquisition from sensors and ERP.
2. **Ingestion and ETL:** Data validated, filtered, transformed.
3. **Storage:** Clean data stored in structured repositories.
4. **Predictive Modeling:** ML models analyze data and generate failure predictions.
5. **Application and Visualization:** Results distributed to applications and dashboards.
6. **Feedback:** User-adjusted results are reintegrated to update models.
7. **Continuous Improvement Cycle:** Self-adjusting, continuously learning and optimizing system.

Colors:

- Yellow: Ingestion and Processing.
- Green: Storage and Feedback.
- Blue: Modeling and Results.
- Orange: Application and Visualization.

3.6 Applied Systems Engineering Principles

1. **Modularity:** Each layer performs an independent function, allowing maintenance and scalability.
2. **Interoperability:** Modules communicate through standardized interfaces (REST APIs) for integration with ERP, sensors, and databases.
3. **Adaptive Feedback:** Visualization information returns to the database to optimize predictive models.
4. **Traceability:** Complete record from data origin to prediction ensures control and transparency.
5. **Functional Hierarchy:** X–Y coordinates visualize the operational hierarchy from capture to result interpretation.

3.7 Architectural Design Conclusion

The proposed architecture integrates a structured and dynamic vision, combining quantitative diagrams (table and bars) with functional diagrams (sequence and pie). This demonstrates a coherent, adaptable, and self-learning system capable of managing complex information and improving performance through constant feedback, complying with systems engineering principles of design, integration, and control.

4 Addressing Sensitivity and Chaos

- Explain how your design addresses highly sensitive variables or chaotic factors (for example, feedback loops, random or unexpected data variations).
- If relevant, propose monitoring routines or error handling for unanticipated conditions.

The system operates stably, self-adjusting, and tolerates disorder thanks to adaptive feedback and its continuous learning loop, which allows us to optimize prediction models.

4.1 Identification of Variables

To verify this, we will first analyze the critical variables:

1. **Component Age:** Exponential increase in failure probability with new problems.
2. **Opening Temperature:** Critical threshold effects on component lifespan.
3. **Usage Intensity:** Non-linear relationship with component wear and cumulative stress effects.
4. **Manufacturing Batch Quality:** Significant impact on baseline reliability.

4.2 Moderation and Control

We know that data quality greatly impacts the technical aspects of the system. Noise, missing values, or inconsistent values pose negative limitations. Furthermore, more sensitive variables such as age, intensity of use, or temperature can generate atypical results. For this, we can implement:

- **Adaptive ETL:** A process that can detect and filter out these extreme values. This way, we can clean and validate the data.

But what happens if there is high commitment? What if unnecessary maintenance is performed? These factors present new challenges for model interpretation. We can avoid this by:

- **L2/L1 Regularization:** Using these regularization techniques is a practical way to reduce overfitting while maintaining data quality and interpretability.

In cases of uncertainty, chaos, or where complex patterns cannot be recognized at all, deep learning techniques and real-time adaptive algorithms must be applied to continuously update the models:

- **Hybrid deterministic–probabilistic models:** Combining deterministic models with probabilistic approaches.

4.3 Continuous and Controlled Feedback

As previously mentioned, the idea of addressing these chaotic factors was primarily based on self-learning and adaptive models. The feedback loops are shown below:

- Failure predictions → Manufacturing improvements → Reduced failure rates
- Service data → Model refinement → Better predictions
- Customer complaints → Quality control adjustments → Product improvements

Considering that the system also needs to be maintained over time by focusing on its performance through constant monitoring and optimization, as well as using improvement guidelines:

- Continuous model retraining with new data.
- Performance monitoring and optimization.
- System updates and security updates.
- Stakeholder feedback integration.

Without forgetting that these models must be progressively tested and evaluated:

- Production system deployment.
- Model performance monitoring.
- Gradual rollout to different business units.
- User training and change management.

4.4 Monitoring

If the system fails, despite all the above considerations, we must ensure that it remains operational even if an essential component or part fails. This is how we can mitigate any unforeseen events:

- Graceful degradation mechanisms for system reliability.
- Self-healing protocols.
- Maintaining copies of data and models.
- Defining tolerance thresholds.

5 Technical Stack and Implementation Plan

This section aims to establish the technical and methodological guidelines necessary for the implementation of the ASUS component failure prediction system. The goal is to translate the conceptual architecture and sensitivity and chaos control strategies into a practical proposal, selecting tools, languages, and frameworks that ensure traceability, scalability, reliability, and interpretability of the system.

This analysis also defines the implementation phases, applied design patterns, quality and security criteria, and minimum resources required to deploy the system into a stable and reproducible operational environment.

5.1 Guiding Principles for Technology Selection

The selection of the technology stack is based on the following principles, aligned with the functional and non-functional requirements defined in previous sections:

- **Scalability:** The system's capacity to process large volumes of information and support the incorporation of new data without performance loss.
- **Traceability:** Tracking processes, models, and data transformations to guarantee reproducibility.
- **Interoperability:** Seamless integration with external sources such as ERP systems and corporate databases.
- **Automation:** Reducing manual tasks through automated ingestion, modeling, and monitoring workflows.

- **Observability:** Implementing metrics and alerts to identify failures or deviations in performance.
- **Fault Tolerance:** The capacity to continue operating despite partial interruptions in sources or modules.

Each principle directly responds to the findings of points 1 to 4, ensuring consistency between conceptual design and technical execution.

5.2 Proposed Technical Stack (by Layer)

The technical stack of the ASUS component fault prediction system is structured into functional layers that maintain a coherent, orderly, and traceable information flow. Each layer performs a specific function within the data analysis and machine learning process, ensuring that data is managed efficiently from its origin to the final visualization of results. The main functions of each component are described below, focusing on academic implementation.

5.2.1 Data Acquisition or Ingestion

This layer is responsible for collecting data from various sources, such as historical maintenance records, sensors, or corporate databases. Its objective is to ensure that information reaches the system continuously, accurately, and securely. In academic environments, Python together with libraries like `pandas` and `requests` is used to perform practical data acquisition from local databases or CSV/Excel files.

5.2.2 Data Processing and Cleaning (ETL)

This stage transforms raw data into clean and structured information, performing validation, removal of inconsistencies, and value normalization. It is essential to guarantee data quality and prevent biases or errors in predictive models. In academic settings, Jupyter Notebook with Python libraries such as `pandas`, `numpy`, and `scikit-learn` is used to develop interactive and reproducible ETL processes.

5.2.3 Data Storage

The storage function is to preserve data in a structured, secure, and accessible manner. Different storage levels are established: original data (raw), processed data (curated), and data used in predictive models (feature store). In academic contexts, relational databases such as MySQL or PostgreSQL are used, with SQLite as a lightweight alternative for prototypes.

5.2.4 Predictive Modeling

This layer constitutes the system's core, where machine learning algorithms are developed, trained, and validated to predict potential failures in ASUS components. In academic contexts, Python with libraries such as `scikit-learn` and `XGBoost` is used to implement interpretable classification models suitable for experimentation and learning.

5.2.5 Result Exposure and APIs

This layer provides users or other systems access to model predictions, ensuring fast, secure, and controlled result delivery. In academic settings, Flask is used to create lightweight and functional APIs to expose model outputs.

5.2.6 Visualization and User Experience

Visualization translates model results into understandable information, facilitating decision-making through dashboards, charts, and interactive reports. In academic contexts, tools such as Power BI Desktop, Tableau Public, or Python's Streamlit library are used to build intuitive dashboards.

5.2.7 Observability and Monitoring

This layer focuses on continuous monitoring of system performance, detecting variations in model accuracy, execution errors, or data pattern changes (data drift). In academic settings, MLflow is used to manually record metrics, document model versions, training parameters, and evaluation results.

5.2.8 Security and Compliance

Security is a transversal component ensuring data integrity, confidentiality, availability, and compliance with internal policies. In academic environments, basic measures such as SSL/TLS encryption, secure database passwords, and user access permissions are applied.

5.3 Applied Design Patterns

The ASUS fault prediction system is structured using design patterns to ensure organization, scalability, and maintainability of the information flow. These patterns help the system remain stable under data variability and facilitate adaptation in both academic and professional contexts.

Key patterns and their functions:

-ETL (Extract, Transform, Load): organizes data flow into extraction, transformation, and loading stages, ensuring consistency and traceability. Recommended software: Python and `pandas` for academic use; Airflow and Spark for professional environments.

-Layered Architecture: separates the system into independent layers —ingestion, processing, modeling, visualization— to facilitate maintenance and integration. Recommended software: scripts or notebooks for education; Flask/FastAPI and Docker containers for professional use.

-MVC (Model–View–Controller): separates model logic, visual presentation, and user interaction control to ensure an adaptable interface. Recommended software: Flask or Streamlit in academic contexts; FastAPI or Django professionally.

-Data Pipeline: automates the sequence of steps moving data from origin to predictive model while maintaining reproducibility. Recommended software: sequential scripts in education; Airflow or Kubeflow in professional systems.

-Observer / Publish–Subscribe (Pub/Sub): enables asynchronous communication between components, triggering alerts or automatic actions. Recommended software: MLflow or local logs academically; Kafka, Prometheus, and Grafana professionally.

-Modularity and Reusability: allows each component —ETL, model, API, visualization— to be developed and replaced independently without affecting others. Recommended software: organized scripts in education; Docker and Kubernetes in enterprise environments.

Together, these patterns create a controlled, transparent, and scalable data flow. The academic version prioritizes understanding and experimentation, while the professional version emphasizes robustness, automation, and real-world applicability.

5.4 Implementation Plan

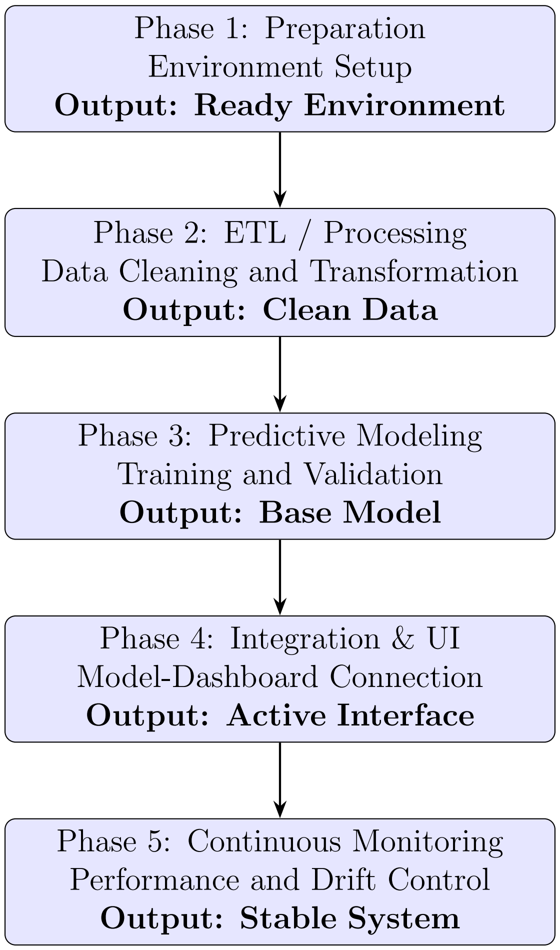

Phase 1: Preparation
Environment Setup
Output: Ready Environment
Phase 2: ETL / Processing
Data Cleaning and Transformation
Output: Clean Data
Phase 3: Predictive Modeling
Training and Validation
Output: Base Model
Phase 4: Integration & UI
Model-Dashboard Connection
Output: Active Interface
Phase 5: Continuous Monitoring
Performance and Drift Control
Output: Stable System

Figure 6: Implementation Workflow of the ASUS Component Failure Prediction System

5.5 Integration with Point 4 (Monitoring and Control)

Integration of the technical design with the monitoring and control strategies from Point 4 ensures system stability and resilience under chaotic conditions or unexpected data variations. Automatic monitoring mechanisms detect deviations in model performance and trigger adjustment or retraining procedures when necessary.

The system employs a continuous observe–evaluate–act flow, where performance metrics are recorded and analyzed in real time. When anomalies are detected —such as increased prediction error or abrupt changes in data distribution— verification routines are executed, and if applicable, model retraining is initiated.

Recommended software:
In academic settings, monitoring can be performed using MLflow, `pandas`, and automated

metric logs.
In professional applications, this process is enhanced using Prometheus, Grafana, and Airflow, allowing visual alerts, historical analysis, and automated responses to unforeseen conditions.

This coupling between technical components and control routines ensures system robustness and maintains performance even in highly variable environments.

5.6 Testing, Quality, and Security

To ensure system reliability, verification, validation, and quality control procedures are defined throughout the development cycle. Each implementation phase undergoes unit, integration, and performance testing, ensuring modules function correctly both individually and collectively.

Additionally, basic security measures and access control are included to protect data integrity and generated results. This includes file encryption, responsible credential management, and input validation during ingestion processes.

Recommended software:
In academic contexts, tests can be carried out using Python native tools (`unittest`, `pytest`) and version control in GitHub.
Professionally, automated continuous validation (CI/CD) systems using GitLab, Jenkins, or Azure DevOps are incorporated, along with security audits and vulnerability scanning.

These strategies guarantee that the system meets minimum quality, security, and reproducibility standards.

5.7 Resources and Team

System development requires an interdisciplinary team with expertise in data analysis, software engineering, and project management. Key roles include:

- **Data Analyst:** responsible for processing, cleaning, and preparing datasets.
- **Software Engineer:** responsible for implementing modules, APIs, and the overall system flow.
- **Data Scientist:** responsible for predictive modeling and results validation.
- **Technical Supervisor or Instructor:** guides the methodological process and ensures academic quality.

In educational settings, infrastructure may be limited to local environments and free tools; in professional environments, the use of virtual servers or cloud platforms such as AWS, Azure, or Google Cloud is recommended for efficient process scaling.

5.8 Risks and Mitigations

Risk	**Description**	**Mitigation Strategy**
Model Drift	Changes in data distribution reducing model accuracy	Continuous monitoring and automated retraining
Data Ingestion Failures	Connection loss or errors in data sources	Pre-validations and temporary backup storage
Integration Errors	Incompatibility between modules or library versions	Unit testing and dependency control with virtual environments
Resource Limitations	Insufficient capacity for processing large volumes	Use representative samples or cloud scaling

Table 1: Main technical and operational risks with mitigation strategies.

5.9 Diagram

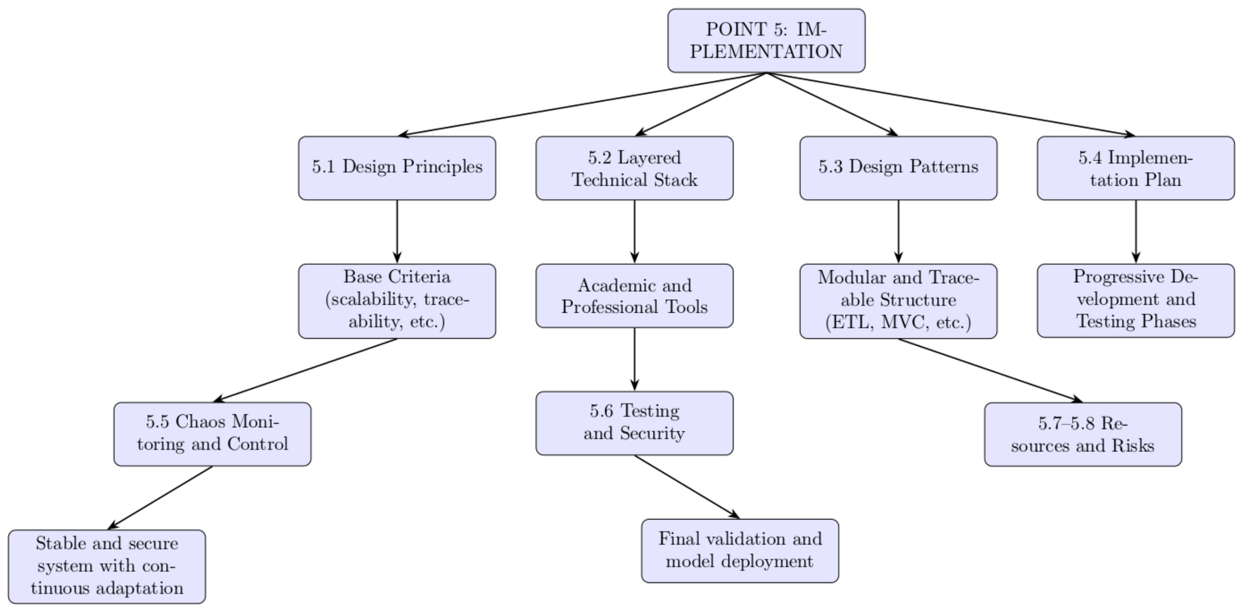

POINT 5: IMPLEMENTATION
5.1 Design Principles
5.2 Layered Technical Stack
5.3 Design Patterns
5.4 Implementation Plan
Base Criteria (scalability, traceability, etc.)
Academic and Professional Tools
Modular and Traceable Structure (ETL, MVC, etc.)
Progressive Development and Testing Phases
5.5 Chaos Monitoring and Control
5.6 Testing and Security
5.7–5.8 Resources and Risks
Stable and secure system with continuous adaptation
Final validation and model deployment

Figure 7: Hierarchical Diagram of Implementation Plan (Point 5)

6 Conclusions

Overall, the analysis developed throughout the five points allowed for the design of a comprehensive system, grounded in systems engineering principles and modern predictive methodologies. The project successfully combines theory and practice, proposing a solution adaptable to different levels of application. It is concluded that the ASUS failure prediction system not only meets the academic objectives of the workshop but also has the potential to evolve into a functional and industrially valuable tool, capable of improving operational reliability through predictive analysis and adaptive control.

6.1 Review of Workshop 1 Findings

The analysis of previous results enabled a deeper understanding of the complex and dynamic nature of the ASUS component failure prediction system. It was evident that component behavior exhibits chaotic characteristics and is dependent on multiple external factors, justifying the need for a systems engineering-based approach. This point served as a conceptual foundation for the entire project, establishing the relationship between chaos theory principles, model sensitivity, and expected operational reliability.

6.2 Definition of System Requirements

A clear identification of functional and non-functional requirements allowed structuring a system that responds to both technical and maintenance management needs. The adopted approach promoted traceability and continuous validation of each requirement against system objectives, ensuring coherence between theoretical design and future implementation. As a result, there is a solid foundation for technical planning and scalable development of the predictive system.

6.3 High-Level Architecture

The proposed layered architecture proved to be an effective solution for organizing data flows and defining precise responsibilities among modules. This modular design, grounded in systems engineering principles, facilitates scalability, maintenance, and interoperability between components. It is concluded that the defined architecture allows integration of different system functions in an orderly manner, ensuring adaptability to both academic and professional environments.

6.4 Sensitivity, Chaos, and Control

The study of sensitivity and the application of chaos theory highlighted the need to implement continuous monitoring mechanisms and adaptive control. It was determined that prediction accuracy may vary with small perturbations, thus strategies were incorporated to mitigate the effects of noise and fluctuations. In conclusion, the inclusion of feedback routines, dynamic metrics, and retraining processes strengthens system stability under uncertain environments.

6.5 Technical Stack and Implementation Plan

The development of the technical plan consolidated design decisions and the selection of appropriate tools for system construction. The dual approach —academic and professional— allowed balancing accessibility and robustness, facilitating implementation in both educational and real-world environments. Moreover, the organization by phases, applied design patterns, and testing strategies ensure a sustainable, scalable, and secure structure for future system development.

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