Workshop No.3 — Systems Engineering Analysis

COVID-19 Global Forecasting

Kaggle Competition

Santiago Vargas Gomez - 20242020139
David Esteban Sanchez Torres - 20221020093
Dilan Guisseppe Triana Jimenez - 20221020100
Daniel Alejandro Castro - 20242020271
Course: Systems Analysis & Design
Professor: Carlos A. Sierra

November 8, 2025

Abstract

This document presents the third stage of the COVID-19 Global Forecasting System development, focused on robust system design and project management. It integrates principles of quality assurance, risk control, and collaborative planning to strengthen system reliability, scalability, and maintainability. Each section is developed by a team member, addressing architecture refinement, risk analysis, project planning, and system evolution.

COVID-19, Robust Design, Systems Engineering, Risk Management, Project Planning, Data Science

1 Refined System Architecture (Robust Design)

1.1 Overview of the System Architecture

The refined system architecture for the COVID-19 Global Forecasting project builds upon the foundations established in previous workshops, focusing now on robustness, scalability, and modular design. This architecture is designed to ensure that the system can handle data variability, maintain consistency across processes, and facilitate transparent and reproducible experimentation. The system follows a pipeline-oriented structure, where each module performs a specific and independent function, contributing to a unified data flow that ensures accuracy from data ingestion to final predictions.

The design embraces principles from systems engineering, including separation of concerns, fault tolerance, and maintainability. Each stage of the system was refined to reduce redundancy and improve reliability through automated validation and monitoring. Additionally, version control via GitHub and integration with the Kaggle API ensures synchronization between datasets, code, and submissions, making the workflow resilient to human or system errors. The refined structure also supports scalability by allowing new models or preprocessing techniques to be integrated without affecting the system's overall operation.

1.2 System Modules Description

The system is organized into five main modules that interact in a linear but feedback-oriented workflow:

1. Data Ingestion: This module retrieves the official competition datasets (`train.csv`, `test.csv`, and `submission.csv`) using the Kaggle API. Data validation checks are performed to ensure structural integrity, such as column name verification and missing value detection.

2. Preprocessing: Once the data is loaded, the preprocessing module handles cleaning, normalization, and feature transformation. It addresses missing entries, standardizes country and province names, and creates new temporal features (e.g., day index, growth rate) to enhance model performance.

3. Modeling: This core component implements machine learning models such as Random Forest Regressor, Gradient Boosting, and XGBoost. The system uses modular model wrappers, allowing quick testing of new algorithms without altering the main workflow. Cross-validation and hyperparameter tuning are applied to ensure generalization and robustness.

4. Evaluation and Feedback: After training, the models are evaluated on a validation subset. Evaluation metrics (e.g., Mean Absolute Error, Root Mean Squared Error) are computed, and the results are compared with previous iterations. A feedback loop automatically stores model performance data, helping guide improvements in preprocessing or parameter configuration.

5. Submission Handler: The final stage formats the predictions according to Kaggle's specifications. It generates a valid `submission.csv` file containing cumulative forecasts for confirmed cases and fatalities. The Kaggle API is then used to upload predictions and retrieve leaderboard scores, closing the feedback cycle.

1.3 Quality and Robustness Principles Applied

To achieve a robust and sustainable system, several quality management and engineering principles were integrated into the design. The architecture applies modularity to isolate errors and prevent propagation; for example, a failure in preprocessing does not affect the modeling phase. This separation ensures reliability and simplifies debugging. Furthermore, redundancy was introduced through automated backups of datasets and model checkpoints.

Following ISO 9001 quality standards, the design enforces traceability through documentation and version control. CMMI principles were adopted to standardize processes and maintain consistency across development iterations. In addition, Six Sigma concepts were considered to minimize variability and improve model accuracy through continuous process evaluation.

Automation plays a critical role in ensuring robustness. The use of Jupyter Notebooks integrated with the Kaggle API automates data retrieval, validation, and result submission. Continuous integration via GitHub allows version tracking and collaborative debugging, ensuring that every modification can be traced and reversed if necessary. This technical foundation not only strengthens the system's reliability but also establishes a scalable and transparent framework adaptable to future prediction challenges.

1.4 Architecture Diagram

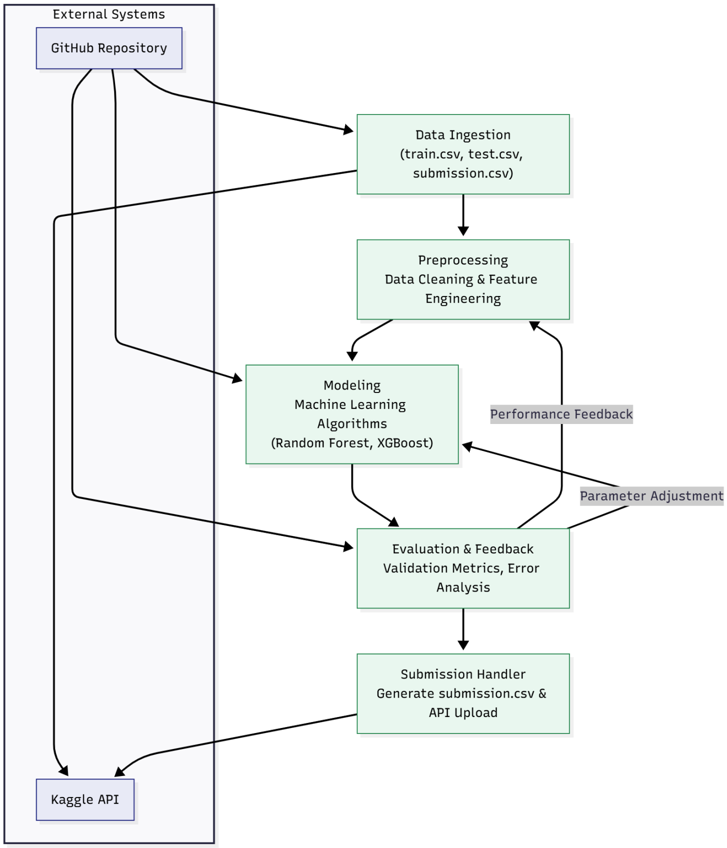

External Systems
GitHub Repository
Data Ingestion (train.csv, test.csv, submission.csv)
Preprocessing Data Cleaning & Feature Engineering
Modeling Machine Learning Algorithms (Random Forest, XGBoost)
Performance Feedback
Parameter Adjustment
Evaluation & Feedback Validation Metrics, Error Analysis
Submission Handler Generate submission.csv & API Upload
Kaggle API

Figure 1: Refined architecture of the COVID-19 Forecasting System showing modular components, feedback mechanisms, and integration with external platforms.

The architecture diagram in Figure 1 illustrates the modular flow of the forecasting system. Data ingestion and preprocessing form the foundation of the pipeline, ensuring data integrity and consistency. The modeling component integrates machine learning algorithms to generate predictions, while the evaluation and feedback stage provides continuous performance monitoring and optimization. External systems such as GitHub and the Kaggle API guarantee traceability, automated submissions, and reproducibility, resulting in a robust and scalable forecasting framework.

2 Quality and Risk Analysis

2.1 Introduction

Ensuring quality and reliability in a forecasting system requires a structured approach to risk management and continuous monitoring. The COVID-19 prediction platform operates in a dynamic environment, where data irregularities, modeling inconsistencies, or system failures can significantly impact performance. Therefore, a robust quality assurance framework was implemented to guarantee consistency, reproducibility, and resilience throughout the workflow. This framework is guided by international quality standards, emphasizing proactive risk identification, mitigation, and traceability of every process involved in the data pipeline.

Quality assurance in this context not only refers to technical performance but also to the methodological integrity of the system. Each process—from data ingestion to model evaluation—was subject to validation protocols, logging mechanisms, and verification steps. The integration of these quality controls reduces uncertainty and establishes a foundation for continuous improvement aligned with the iterative feedback process described in the previous section.

2.2 Identified Risks

Through system analysis and iterative testing, several potential risks were identified that could compromise the functionality or accuracy of the system:

1. Data Loss or Corruption: The use of external APIs (Kaggle) and large datasets introduces a risk of incomplete or corrupted downloads, particularly due to network instability or user errors during synchronization.

2. Model Overfitting: Because of limited data for certain countries or time periods, models may capture noise instead of actual patterns, leading to inaccurate future predictions.

3. API Connection Failure: Dependence on the Kaggle API for dataset retrieval and submission introduces a potential single point of failure if the API becomes temporarily unavailable.

4. Version Mismatch or Configuration Errors: When multiple contributors update code or libraries, there is a possibility of dependency mismatches, causing reproducibility issues or system crashes.

5. **Human Errors in Submission:** Incorrect formatting of the `submission.csv` file or failure to follow competition guidelines could result in disqualified entries.

2.3 Risk Mitigation Strategies

Each identified risk was evaluated according to its probability of occurrence, potential impact on system performance, and mitigation strategy. The following table summarizes these findings:

Table 1: Risk Analysis Matrix

Risk	**Probability**	**Impact**	**Mitigation Strategy**
Data Loss or Corruption	Medium	High	Validate datasets with checksum; automate backups; perform integrity tests before processing.
Model Overfitting	High	Medium	Apply cross-validation, feature regularization, and early stopping to ensure generalization.
API Connection Failure	Medium	High	Implement retry logic, local caching, and alternative manual download procedures.
Version Mismatch	Medium	Medium	Use environment files (`requirements.txt`) and Docker containers to ensure consistent environments.
Submission Errors	Low	High	Automate file generation and add a pre-submission validation script to check column structure.

These mitigation mechanisms were integrated into the development process as part of the continuous quality control cycle. Automated integrity checks ensure that data and model outputs meet predefined conditions before progressing to the next stage, while redundant backups and reproducible environments prevent loss of progress or inconsistency in results.

2.4 Monitoring and Quality Assurance

To sustain reliability and prevent risk recurrence, the system includes continuous monitoring tools and documentation protocols. Error logs and validation results are automatically recorded, allowing for post-mortem analysis in case of failure. GitHub is employed as the central version control and monitoring platform, providing full traceability of code changes, dataset updates, and experiment histories.

A structured review process was established where each new model version or data preprocessing change undergoes peer validation within the team. This ensures that no unverified component reaches the final stage of submission. In addition, regular testing cycles were scheduled to evaluate the stability of the system across different environments and hardware configurations.

The integration of these quality assurance mechanisms reinforces the system's robustness. By combining preventive risk management, process standardization, and collaborative validation, the project maintains high accuracy while minimizing the probability of system degradation. This comprehensive approach ensures that the forecasting pipeline remains transparent, reproducible, and adaptable to future updates or new datasets.

3 Project Management Plan

3.1 Roles and Responsibilities

Team Member	Role	Main responsibilities
David Sanchez	System Architect / Technical Lead	Design the refined architecture, ensure modularity, robustness, and traceability.
Dilan G. Triana	Quality Analyst / Risk Manager	Identify, evaluate and monitor risks; execute mitigation strategies and quality controls.
Daniel A. Castro	Project Planning / Documentation	Organize documentation, plan project tasks, and support communication flow.
Santiago Vargas	Systems Analyst / Integrator	Integrate improvements across modules; ensure consistency between system versions and document implementation contributions.

This assignment guarantees a balance of responsibilities, accountability, and collaboration efficiency.

3.2 Methodology

A **Kanban methodology** was adopted to provide visibility of progress and flexibility in task scheduling.

The workflow is managed through a shared board with the following stages:

- **To Do**
- **In Progress**
- **Review / Validation**
- **Done**

This method supports iterative improvements based on leaderboard feedback and course evaluations.

3.3 Milestones and Deliverables

Milestone	Deliverable	Trget Date	Status
Workshop 1 Preparation	Initial system analysis outline	Sep 22, 2025	Completed
Workshop 1 Submission	System Analysis Report	Sep 27, 2025	Completed
Workshop 2 Preparation	Requirements + architecture refinement	Oct 13, 2025	Completed
Workshop 2 Submission	System Architecture & Requirements	Oct 18, 2025	Completed
Workshop 3 Planning	Robust design + project plan	Nov 3, 2025	Completed
Workshop 3 Submission	Updated architecture + Risk + Project Management Plan	Nov 8, 2025	Completed
Workshop 4 Development	Simulation scenarios implementation	Nov 10 - 28, 2025	Pending
Workshop 4 Submission	Simulation report + GitHub update	Nov 29, 2025	Pending

3.4 Project Timeline

Phase	Duration	Key Activities
Initial Analysis	Sep 15 – Sep 27	Data review, identification of system boundaries
Architecture & Requirements	Sep 28 – Oct 18	Definition of modules, validation, quality constraints
Robust Design & Project Management	Oct 19 – Nov 8	Architecture refinement, risk control, project plan
Simulation Development (Workshop 4)	Nov 10 – Nov 28	ML-based simulation + event-based simulation, validation through system workflow
Final Improvements and Documentation	Nov 29	Integrate simulation results, prepare final delivery, reporting

3.5 Communication and Monitoring

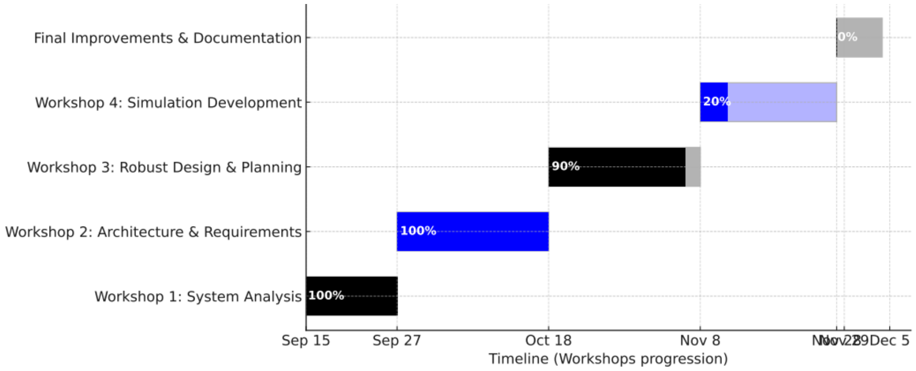
To ensure coordination and progress tracking, the team uses:

- **WhatsApp** → Daily communication and teamwork discussions
- **Google Drive** → Collaborative editing of workshop documents
- **GitHub** → Version control of code and architecture assets
- **Kaggle** → Submission feedback and performance evaluation

Monitoring is conducted through:

- Kanban board task tracking
- Review checkpoints before each workshop submission
- Leaderboard score evaluation to guide model refinement

This structure enables timely deliveries, transparency, and effective collaboration across all modules.



4 Incremental Improvements

4.1 Previous Feedback Summary

During Workshops 1 and 2, several areas for improvement were identified in the early version of the COVID-19 Forecasting System. Initially, the system operated as a basic analytical prototype with limited functionality, relying only on simple modeling and evaluation metrics. Feedback from Workshop 1 emphasized three main weaknesses: the lack of modularity in the architecture, the high sensitivity of the model to noisy and incomplete data, and the absence of validation and feedback mechanisms. These observations highlighted that the system could not ensure data integrity or maintain prediction stability when confronted with inconsistent or missing information. Additionally, the group recognized the need for clearer traceability of data sources, version control, and reproducibility standards to align the project with systems engineering best practices.

4.2 Implemented Improvements

In response to this feedback, the system was restructured into a multi-stage modular pipeline, consisting of data ingestion, preprocessing, modeling, evaluation, and feedback control modules. Each component was redesigned to operate independently yet collaboratively, improving flexibility and maintainability. One of the most significant changes was the integration of a feedback loop connecting the evaluation and modeling stages. This mechanism allows the model to automatically retrain or adjust its parameters when prediction errors exceed a defined threshold, effectively creating a self-correcting and adaptive system.

At the data ingestion level, validation and integrity checks were introduced to detect missing values, duplicated records, and temporal inconsistencies. The preprocessing module was expanded to include normalization, outlier detection, and feature creation, directly addressing the need for data cleansing before model training. Additionally, traceability and reproducibility tools such as metadata logs and data hashing were added, allowing every dataset and model configuration to be tracked precisely. From a technical perspective, improvements also included hyperparameter tuning, ensemble techniques (Random Forest and XGBoost), and GitHub-based version control, which enhanced scalability and teamwork efficiency.

4.3 Impact of the Improvements

The implemented changes had a substantial impact on both system performance and project organization. The modular architecture simplified debugging and experimentation, while the feedback loop introduced adaptability and resilience in the face of data variability. As a result, the forecasting model achieved greater stability and accuracy, especially when handling missing or inconsistent data. Moreover, the integration of data validation and metadata logging improved the traceability and transparency of the entire process, aligning the system with professional standards of reproducibility and auditability.

On the organizational side, using GitHub for version management significantly improved collaboration among team members by reducing conflicts and enhancing workflow synchronization. The group also developed a deeper understanding of how modular design and controlled feedback contribute to the robustness of complex systems. Overall, the project evolved from a static analytical model into a dynamic, maintainable, and self-correcting forecasting system, demonstrating clear alignment between theoretical systems design principles and practical data engineering implementation.

5 Reflection

Working on the COVID-19 Forecasting System taught us that when the data had missing values or errors, the model's results changed a lot (sensitivity). Because of this, we understood that a forecasting system must include steps for checking data quality, detecting problems, and adapting automatically to changes instead of relying on fixed settings.

Using systems engineering principles helped us organize and manage the project in better way by making the system modular, adding feedback loops, and keeping track of data versions made the design easier to understand and maintain.

In terms of teamwork we had problems with file versions and mixed code. Later, we fixed that by using GitHub for version control and dividing responsibilities clearly, allowing everyone to work efficiently.

Finally, we realized that forecasting systems must be flexible and adaptable, especially when the data are uncertain or changing quickly. Our focus on modularity, validation, and feedback helped the system remain stable even when the data was not perfect. Overall, the project helped us to understand how teamwork, system design, and data management come together to build a reliable and intelligent forecasting solution.

References

[1] Kaggle, “COVID-19 Global Forecasting Week 1,” 2020. [Online]. Available: `https://www.kaggle.com/competitions/covid19-global-forecasting-week-1`

[2] ISO 9001:2015, “Quality management systems—Requirements,” International Organization for Standardization, 2015.

[3] CMMI Institute, “Capability Maturity Model Integration,” Version 2.0, 2020. [Online]. Available: `https://cmmiinstitute.com/cmmi`

[4] M. J. Harry and R. Schroeder, *Six Sigma: The Breakthrough Management Strategy Revolutionizing the World’s Top Corporations*. Currency, 2000.