Interactive Footwear Defect Control System: Architecture, Implementation, Real-Time Processing and Multi-Company Operation Based

José de Jesús Sandoval-Palomares
Information Technology Research and Development Department, CIATEC, Mexico

* Corresponding Author: **José de Jesús Sandoval-Palomares**

Article Info

ISSN (Online): 3107-6580
Impact Factor (RSIF): 8.23
Volume: 02
Issue: 04
Received: 10-05-2026
Accepted: 08-06-2026
Published: 06-07-2026
Page No: 61-67

Abstract
Quality control in manufacturing environments, requires robust information systems that integrate distributed defect capture in inspection areas, real-time monitoring, historical analysis, and quality objective management. This article presents "defectos_elink", an interactive web-based system developed in Python that centralizes defect control operations for multiple companies. The system utilizes a server-side rendering architecture with Django, employing MySQL for centralized per-company storage and SQLite as a synchronization mechanism with remote capture stations. A role-based permission model is implemented at both the module and component levels. Operational results demonstrate the system's ability to process real-time events, manage defect and inspector catalogs, synchronize configurations to remote stations via SQLite and MQTT, and maintain complete data isolation between companies through independent MySQL databases. The system is deployed in production using Docker containers with Coolify or a traditional Gunicorn/Nginx architecture on Ubuntu.

DOI: https://doi.org/10.54660/.IJECA.2026.2.4.61-67

Keywords: Django, defect tracking, real-time, synchronization

1. Introduction

The state of Guanajuato —and the city of León in particular—is home to Mexico's primary leather and footwear cluster. The state boasts over 6,200 economic units linked to the sector, approximately 98.9% of which are micro, small, or medium-sized enterprises. Guanajuato accounts for around 65% of national footwear production, exports to 67 international destinations, and contributes approximately 7.5% of the state's Gross Domestic Product [1]. Footwear manufacturing companies in Mexico exhibit varying levels of maturity in their quality control processes; while some utilize ERP systems, spreadsheets, or internal applications, others continue to record defects using printed forms, notes, or isolated reports. This heterogeneity can limit the timely availability of information, the traceability of non-conformities, and the identification of patterns associated with products, operations, materials, machinery, or production lines.

Defect logging relies heavily on manual input and faces the constant challenge of maintaining and improving product quality through the timely detection and correction of defects. Footwear defects—such as excess glue, debonding or weak cementing, wrinkles, and open seams, among others—are particularly difficult to identify automatically due to the wide variety of footwear types and the varying frequency or significance of these issues (see Figure 1). The volume of generated data is substantial, requiring efficient processing, analysis, and visualization to support decision-making.

Traditionally, these quality control systems operate in isolation on the production line; often, due to operational complexity, record-keeping remains manual—hindering the consolidation of information at the corporate level and resulting in delayed, reactive decision-making. There is a clear need to centralize the management of defect catalogs, quality targets, and user productivity monitoring while maintaining the ability to operate across multiple locations without data mixing; this requires a carefully designed architecture. This article presents the "defectos_elink" system—an interactive web-based platform—which addresses these challenges by:

1. A server-side rendering architecture that eliminates dependencies on external frontend frameworks.
2. A granular, role-based authorization model.
3. Multi-tenant support using independent databases.
4. Bidirectional synchronization with remote data capture stations.

Fig 1: Some representative critical defects in footwear

2. Literature review

The digital transformation of quality management is understood as the integration of traditional quality principles with Industry
4.0 technologies—such as connectivity, automation, and data analytics. Reviews by [2], Dias *et al.* [3], and Kumar *et al.* [4] indicate that Quality 4.0 expands the scope of quality management through the systematic use of data, digital traceability, and decision-making support. Bousdekis *et al.* [5] identify data analytics as a core element of quality, given its ability to transform historical and operational data into descriptive, diagnostic, and predictive indicators. Filz *et al.* [6] propose a data-driven quality management platform that integrates heterogeneous production and quality information. The integration between production stations and central systems also aligns with the principles of smart manufacturing. Lee *et al.* [10] propose an architecture in which data obtained at the physical level is processed and communicated to higher levels to support monitoring, benchmarking, and decision-making.

Small and medium-sized enterprises face specific conditions regarding the adoption of technologies. Mittal *et al.* [11] identify barriers such as limited resources, heterogeneous infrastructure, a lack of specialized knowledge, and difficulty justifying complex investments. For this reason, modular architectures, low-cost devices, and web applications can facilitate digitalization. In the leather and footwear industry, a significant portion of technological research has focused on automated surface inspection using computer vision. Aslam *et al.* [12] reviewed computer vision methods used to detect and classify leather defects, while Chen *et al.* [13] analyzed techniques based on image processing and deep learning to inspect surface irregularities.

User administration requires mechanisms that limit the functions and data available to each profile; the role-based access control model, formalized by Sandhu *et al.* [14], assigns permissions based on organizational responsibilities rather than configuring them individually for each user. In systems serving different companies or organizational units, data isolation constitutes another key requirement. Aulbach *et al.* [15] analyzed various strategies for storing information from multiple clients in Software-as-a-Service applications.

The MQTT standard incorporates Quality of Service levels, sessions, acknowledgments, and other mechanisms to control message delivery [16]. Combining a local SQLite database with subsequent publication via MQTT allows a station to continue logging events during a communication outage and synchronize them once the connection is restored.

A review of the literature reveals that existing studies typically address the digitalization of quality, manufacturing analytics, automated inspection, multi-company operations, and communication with remote devices in isolation. The contribution of the proposed system lies in integrating these elements into a platform designed for defect control in footwear manufacturing. Specifically, it combines inspector-assisted data capture, local operation, synchronization with a central server, historical analysis, monitoring, quality targets, alerts, role-based permissions, and data isolation by company.

3. Methodology

This project was conducted as an applied technological study with a descriptive scope, employing a case study approach. The subject of the study was the design, implementation, and functional validation of an information system for recording, monitoring, and analyzing defects in footwear manufacturing processes. The methodology focused on addressing operational needs related to the distributed capture of defects, data recording continuity during communication failures, centralized information consolidation, and role-based access.

3.1. Requirements Analysis

In the first stage, the activities performed by inspectors, supervisors, managers, directors, and administrators within the defect control process were analyzed. The following were identified as key requirements:

- Recording defects directly from stations located on production lines.
- Maintaining local operations when there is no communication with the central server.
- Transmitting recorded events to a central platform.
- Querying defect data in real-time and historically.
- Managing catalogs of defects, inspectors, stations, lines, and departments.
- Defining quality objectives by defect type and department.
- Generating indicators, Pareto charts, tables, and alerts.
- Restricting access to modules and components based on each user's role.

Based on these requirements, two main components were defined: a capture application installed on a touchscreen station and a central application for administration, analysis, and monitoring. The remote station allows the inspector to select defects and view recorded entries, while the central application provides concurrent access to various modules based on each user's assigned role.

3.2. Architecture Design

A distributed client-server architecture was adopted, consisting of local data capture stations, a communication service, and a central application. The stations operate using an SQLite database, which serves as temporary storage and ensures continuity when there is no connection to the server. The central application was implemented using Python 3.12.10, Django 6.0.5, and MySQL 8.4.0; Django was used to manage authentication, sessions, forms, views, HTML templates, and administrative functions. MySQL served as the central storage for operational records, catalogs, and user data.

3.3. Data Modeling

The data model was designed to represent the key entities involved in defect control; relevant entities include companies, plants, production lines, departments, stations, defects, users, and roles. Each defect event contains sufficient information to identify the recorded defect, the originating station, the inspector, the line, the department, and the date and time of the record. When the event is generated at a remote station, it is first stored locally and subsequently transmitted to the central system.

3.4. Iterative system development

The system was built using an incremental process. In each iteration, a set of functionalities was implemented, its operation verified, and it was subsequently integrated with existing modules. The first iteration focused on capturing defects from the stations. Subsequently, functions for querying, monitoring, catalog management, quality targets, user management, and alert generation were developed. Communication between remote stations and the central server was implemented using MQTT based on a publish-subscribe model. Each station publishes generated events to a topic configured for the corresponding company, plant, or line. This mechanism aims to prevent a communication interruption from blocking defect registration. Local data persistence allows the station to continue operating and enables data transmission once connectivity is restored.

3.5. Authentication, authorization, and data isolation

Authentication was implemented using Django's user and HTTP session system. The defined roles were administrator, manager, supervisor, inspector, and director. Inspectors focus on defect logging; supervisors and managers access operational and analytical functions; directors primarily have access to query functions; and administrators manage users, permissions, and configuration.

Data isolation between companies was achieved through independent MySQL databases. Furthermore, connections, credentials, MQTT topics, and configuration variables were managed by environment rather than being hardcoded into the source code.

4. System Architecture

4.1. Technology Stack

The "defectos_elink" system is built on Django, using Python as the primary language. Django was chosen due to its maturity as a web framework, its built-in authentication system, its robust ORM, and its integrated administration panel. The complete stack comprises:

Backend: Django + Django REST Framework
Operational database: MySQL
Local storage per station: SQLite
Visualization: matplotlib
Production server: gunicorn
Proxy: nginx (on Ubuntu) / Coolify (on Docker)
MQTT: Communication with remote stations

4.2. Layered Architecture

The system is organized into four functional layers:

Presentation layer: HTML templates with inline CSS, fully server-rendered.

Business logic layer: Django function-based views that delegate heavy logic to Python services. The `services.py` file centralizes all interaction with MySQL using direct mysql.connector calls.

Data layer: MySQL as central storage (Django tables + business tables), SQLite as a publishing layer for remote stations.

4.3. Authorization Model

The system implements a four-layer authorization model evaluated in strict order:

Global switch that enables or disables an entire module for all users.
Role-based permission to access a specific module.
Global switch for internal components (charts, tables, filters, tabs).
Role-based permission to view specific components within a module.

This design allows for granular control: an administrator can globally disable a chart or tab without modifying individual roles, and a role can have restricted access to certain components within a module to which it otherwise has access.

- **Five functional roles are defined:**

Admin: Full control, access to the Django admin interface.
Manager: Operational management of defect monitoring. Defines catalogs, objectives, analyses, and alerts.
Supervisor: Operational queries and monitoring.
Quality Inspector: Enters defects at a production line station.
Director: Read-only access to tables and charts.

4.4. Multi-company or multi-location architecture

Each company operates with its own independent MySQL database, ensuring complete data isolation. Configuration is handled via a `config.ini` file during development, and through environment variables for production in Coolify/Docker.

The database profile switching mechanism defaults to using the `[mysql]` section for standard operations (SELECT, INSERT, UPDATE, DELETE).

5. Implementation

5.1. Project Structure

The defectos_elink app is organized as a project comprising two applications:

The first is installed on a Raspberry Pi-based touchscreen station and is used by quality inspectors to log defects (Figure 2). The second is the central application for analyzing and viewing logged defects in real time; it resides on a central server and supports multi-user access, with menu options for specific modules appearing based on the user's assigned role (Figure 3).

Fig 2: Station screen, used by quality inspectors.

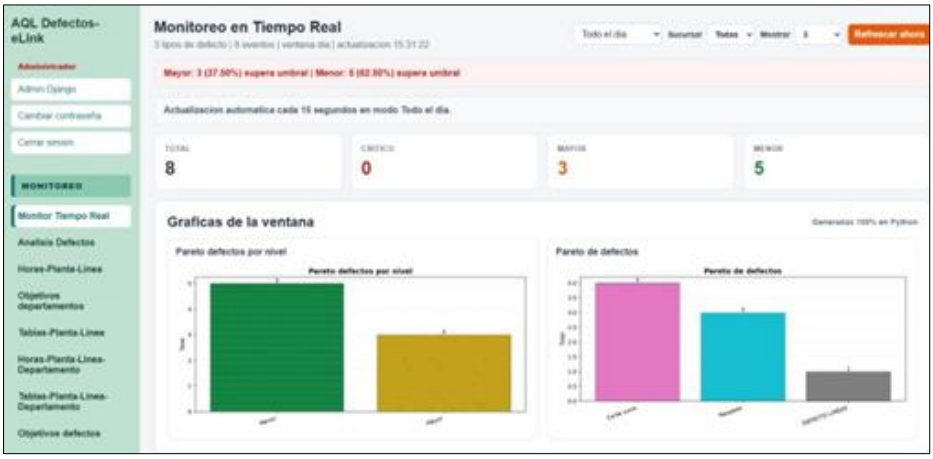


Fig 3: Main screen and options menu of the web platform

5.2. Functional Modules

At the remote station, the only option is to display defects; selecting one generates a record and displays a table of the recorded defects.

The system implements 17 functional modules grouped into three sections of the side menu:

MONITORING Section (8 modules):

- **Real-Time Monitor:** KPIs, Pareto charts, and tables with programmable auto-refresh.
- **Defect Analysis:** Historical data query with 7 filters, 7 Pareto charts, and 3 detailed tables.
- **Plant-Line-Hour:** Defects aggregated by hour.
- **Plant-Line-Hour-Department:** Departmental breakdown of hourly aggregates.
- **Plant-Line Tables:** Defect tables by plant and line.
- **Plant-Line-Department Tables:** Tables with checkbox-based visibility controls.
- **Department Targets:** Monitoring of quality targets by department.
- **Defect Targets:** Monitoring of quality targets by specific defect.

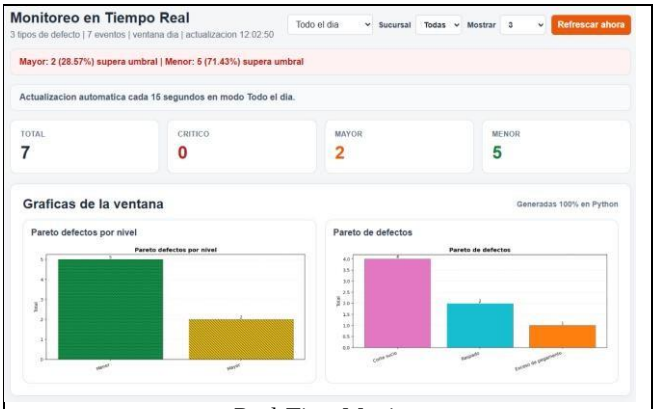
MANAGEMENT Section (8 modules):

- **Defect Reassignment:** Assignment and review of defects by department.
- **Defect Catalog:** Management of defects, profiles, and stations; SQLite synchronization to stations.
- **Defect Quality Targets:** Targets for individual defects.
- **Department Quality Targets:** Targets by department.
- **Inspector Catalog:** Registration, assignment, and synchronization of inspectors to stations.
- **Inspector Users:** Creation of inspectors from Django users.
- **Users and Access:** Full administration of users and operational roles.

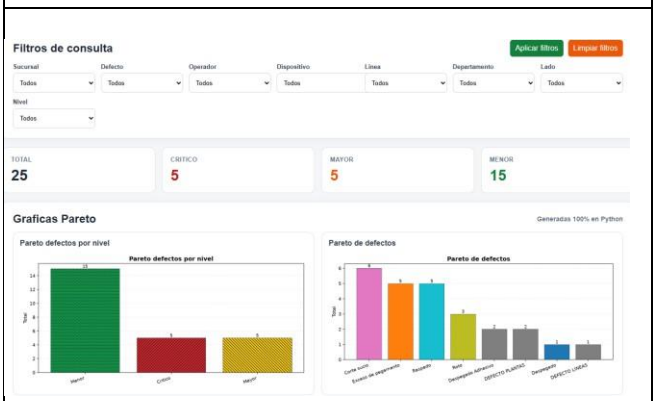
ALERTS Section (1 module):

Real-Time Alerts: Configurable alert rules with real-time evaluation.

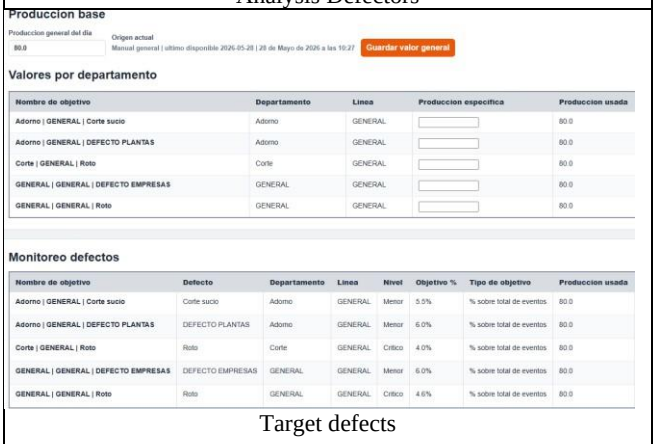
Representative images of these sections are attached (Figure 4).



Real-Time Monitor

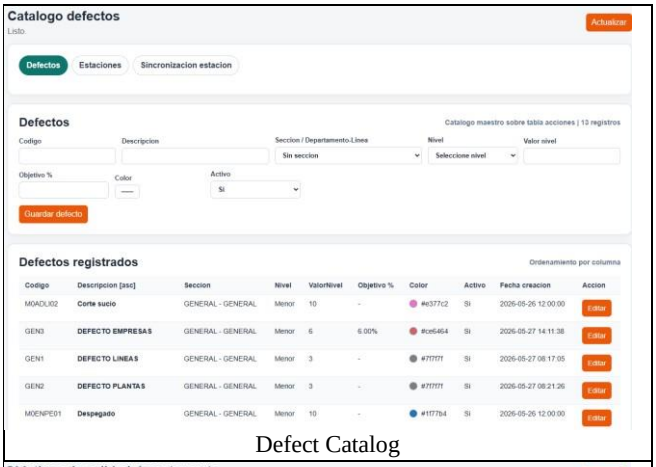


Analysis Defectors

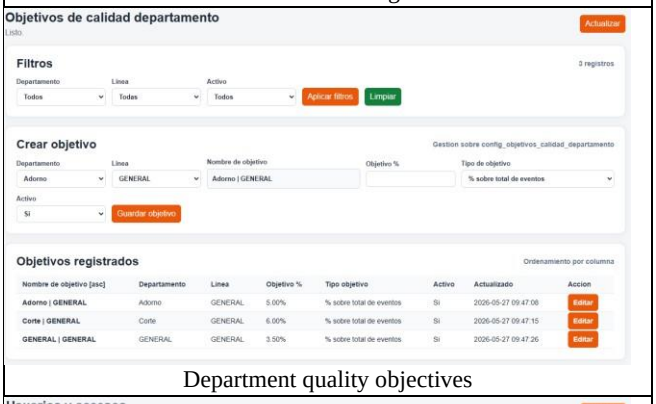


Target defects

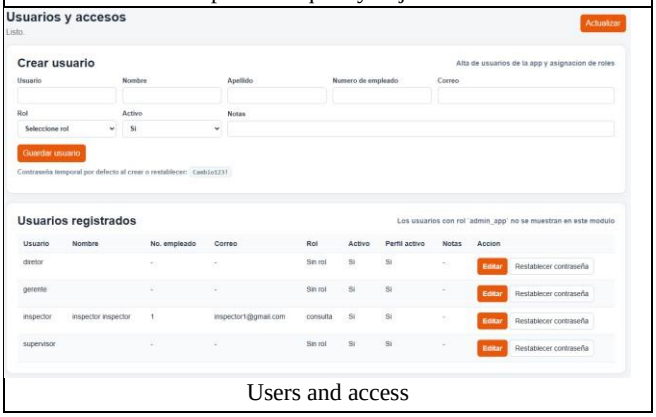
Tables-Plant-Line



Defect Catalog



Department quality objectives



Users and access

Fig 4: Representative images of the web platform

5.3. Deployment

The system supports three deployment modes (Figure 5): **Development (Windows):** runserver or uvicorn on port 8002.
Pre-production (Ubuntu): gunicorn + systemd + nginx.
Production (Ubuntu-Docker + Coolify): Container with gunicorn, proxy managed by Coolify.

Security is implemented across multiple layers:

- **Authentication:** Django Auth with HTTP sessions.
- **Authorization:** Proprietary four-layer model described in section 3.3.
- **Database:** SQLite is used at the stations; MySQL is used for the central system.
- **Event transmission:** Events are sent from the station to a topic via MQTT to prevent data loss due to communication failures.
- **HTTPS:** Configurable SSL proxy header. Standard Django protection

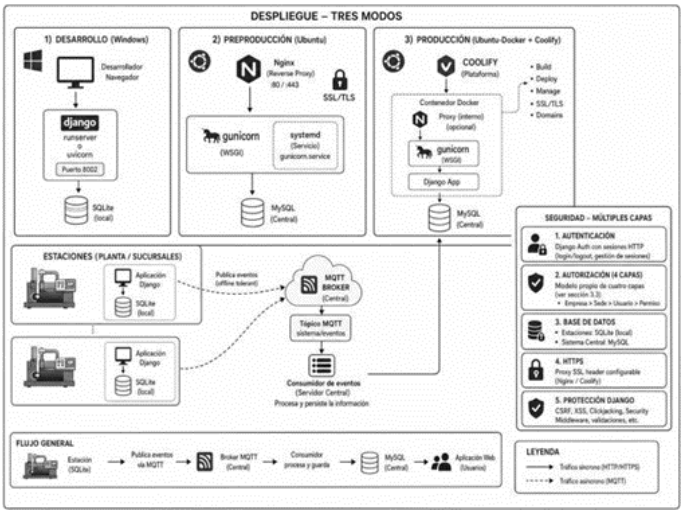


Fig 5: Deployment of the web platform for real-time defect processing

6. Discussion

The choice of plain Django (server-side rendering) is driven by several factors: operational simplicity—eliminating the need to build and deploy a separate frontend; consistency in chart generation by centralizing it on the server; a reduced attack surface due to fewer JavaScript-based dependencies; and ease of maintaining roles and frontend functionality. It also offers model-level validation, managed relationships, controlled migrations, and automated administration tools.

7. Conclusion and Future Work

This article presents the *defectos_elink system, an interactive defect control system based on Django that integrates real-time monitoring, historical analysis, catalog management, quality targets, configurable alerts, and synchronization with remote stations using SQLite and MQTT. The server-side rendering architecture, the four-layer authorization model, and multi-tenant support via independent MySQL databases constitute the system's main contributions.

Future work includes:

1. Refactoring into domain-specific modules to improve maintainability.
2. Implementation of a formal REST API with an OpenAPI specification.
3. Integration of BI tools for predictive defect analysis.

References

1. Gobierno del Estado de Guanajuato. Guanajuato lanza “Pisando Firme”, estrategia que protege y potencia la industria del calzado. Boletines Dependencias. 2025 Sep 18. Available from: https://boletines.guanajuato.gob.mx/
2. Chiarini A. Industry 4.0, quality management and TQM world: a systematic literature review and a proposed agenda for further research. TQM J. 2020;32(4):603-616. doi:10.1108/TQM-04-2020-0082.
3. Dias AM, Carvalho AM, Sampaio P. Quality 4.0: literature review analysis, definition and impacts of the [incomplete reference].
4. Bousdekis A, Lepenioti K, Apostolou D, Mentzas G. Data analytics in Quality 4.0: literature review and future research directions. Int J Comput Integr Manuf. 2023;36(5):678-701. doi:10.1080/0951192X.2022.2128219.
5. Filz MA, Bosse JP, Herrmann C. Digitalization platform for data-driven quality management in multi-stage manufacturing systems. J Intell Manuf. 2024;35:2699-2718. doi:10.1007/s10845-023-02162-9.
6. Lee J, Bagheri B, Kao HA. A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. Manuf Lett. 2015;3:18-23. doi:10.1016/j.mfglet.2014.12.001.
7. Mittal S, Khan MA, Romero D, Wuest T. A critical review of smart manufacturing and Industry 4.0 maturity models: implications for small and medium-sized enterprises. J Manuf Syst. 2018;49:194-214. doi:10.1016/j.jmsy.2018.10.005.
8. Aslam M, Khan TM, Naqvi SS, Holmes G, Naffa R. On the application of automated machine vision for leather defect inspection and grading: a survey. IEEE Access. 2019;7:176065-176086. doi:10.1109/ACCESS.2019.2957427.
9. Chen Z, Deng J, Zhu Q, Wang H, Chen Y. A systematic review of machine-vision-based leather surface defect inspection. Electronics. 2022;11(15):2383. doi:10.3390/electronics11152383.
10. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. Computer. 1996;29(2):38-47. doi:10.1109/2.485845.
11. Aulbach S, Grust T, Jacobs D, Kemper A, Rittinger J. Multi-tenant databases for software as a service: schema-mapping techniques. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. New York: ACM; 2008. p.1195-1206. doi:10.1145/1376616.1376736.
12. Banks E, Briggs E, Borgendale K, Gupta R, editors. MQTT Version 5.0. OASIS Standard; 2019 Mar 7.

How to Cite This Article

Sandoval-Palomares JJ. Interactive footwear defect control system: architecture, implementation, real-time processing and multi-company operation based. Int J Eng Comput Appl. 2026;2(4):61-67. doi:10.54660/.IJECA.2026.2.4.61-67.