ISSN 2278-3091
Volume 15, No.4, July – August 2026
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse011542026.pdf
https://doi.org/10.30534/ijatcse/2026/011542026

# Design and Implementation of the Digital Quality Intelligence Platform: A Modular Decision-Support Software Framework for Engineering Quality Intelligence

**Janice Benita F[1]**
[1]Department of Information Technology, St. Joseph's College of Engineering (Autonomous), Chennai, India, janicebenita123@gmail.com

Received Date: May 12, 2026 Accepted Date: July 18, 2026 Published Date: August 06, 2026

## ABSTRACT

Engineering organizations progressively generate large volumes of heterogeneous quality data through inspection records, laboratory tests, production logs, non-conformance reports, statistical audits, and project documentation. However, these data are commonly processed through disconnected spreadsheets, isolated statistical utilities, and manually interpreted dashboards. Such fragmented workflows weaken traceability, delay corrective action, and limit the reuse of engineering knowledge across projects. This paper presents the design and implementation of the Digital Quality Intelligence Platform (DQIP), a modular decision-support software framework for engineering quality intelligence. DQIP integrates data ingestion, preprocessing, statistical process control, Six Sigma quality assessment, rule-based reasoning, visualization, reporting, and recommendation generation within a layered software architecture. The framework separates domain-independent services from configurable domain knowledge so that the same platform can support multiple engineering quality scenarios without redesigning the core application. The proposed architecture is organized into presentation, application, analytics, reasoning, data management, and infrastructure layers, with service interfaces that support maintainability, extensibility, and independent module evolution. A prototype implementation using Python-based open-source components demonstrates the feasibility of combining engineering analytics and explainable decision support in a lightweight deployable platform. The evaluation focuses on functional adequacy, modularity, maintainability, responsiveness, interoperability, and cross-domain adaptability. Results indicate that DQIP offers a coherent software engineering solution for transforming raw engineering quality records into actionable intelligence while reducing manual workflow dependency. The platform establishes a reusable foundation for future integration with predictive models, digital twins, cloud services, and enterprise quality-management systems.

**Key words:** Decision Support System, Software Architecture, Quality Intelligence, Engineering Analytics, Modular Software Design, Statistical Process Control, Six Sigma Analytics, Engineering Informatics

## 1. INTRODUCTION

Digital transformation has changed the way engineering organizations plan, monitor, and evaluate quality. In construction, manufacturing, infrastructure, energy, and process industries, quality-related information is no longer restricted to final inspection documents. It is now produced continuously by laboratory instruments, inspection teams, production systems, enterprise resource planning software, sensor networks, and document-management repositories. These data streams can support early-defect detection, evidence-based decision-making, and organizational learning. However, their value depends on the availability of software systems capable of transforming distributed records into reliable engineering intelligence.

Many engineering organizations still manage quality data through fragmented software practices. Test results may be stored in spreadsheets, inspection observations may remain in PDF reports, statistical analysis may be performed in independent desktop tools, and corrective-action decisions may be documented separately. This pattern leads to repeated data entry, inconsistent formulas, weak version control, poor traceability, and delays in interpreting non-conformances. The problem is not merely the absence of analytical algorithms. It is the absence of coherent software architecture that integrates data acquisition, statistical evaluation, knowledge representation, visualization, and decision support within a maintainable engineering platform.

Software engineering research has repeatedly shown that the long-term usefulness of an information system depends on architectural qualities such as modularity, interoperability, scalability, testability, usability, and maintainability [1]-[7]. These concerns are especially important in engineering quality management because quality rules evolve across standards, projects, materials, clients, and operating environments. A tool developed for one project may become ineffective when the input schema, acceptance thresholds, reporting method, or

analytical procedure changes. Therefore, engineering decision-support software must be designed as a configurable and extensible platform rather than a fixed single-purpose application.

Decision-support systems provide a useful foundation for addressing this challenge. Classical decision-support research emphasizes the combination of data, models, and user interaction to improve semi-structured decision-making [11]-[13]. In engineering quality management, decisions are often semi-structured: the organization may have statistical evidence, tolerance limits, specifications, and historical records, but an expert must interpret whether the process is stable, whether corrective action is required, and whether the available evidence is sufficient. A software platform that combines statistical analytics with transparent rule-based reasoning can therefore improve both consistency and explainability.

The need for such platforms has increased with Industry 4.0, smart manufacturing, and digital twin initiatives, where data-intensive engineering processes are expected to provide near real-time feedback and support automated decision loops [20]-[24]. Nevertheless, many available systems remain either domain-specific or analytics-specific. Enterprise quality-management systems support documentation and workflow control but are often less flexible for custom engineering analytics. Statistical process control applications provide charts and capability indices but rarely provide end-to-end decision reasoning. Dashboard tools offer visualization but typically require separate data preparation and manual interpretation. Artificial intelligence solutions may improve prediction but often introduce black-box behavior and additional maintenance concerns if they are not embedded within disciplined software architecture.

This paper addresses these limitations by proposing the Digital Quality Intelligence Platform (DQIP), a modular decision-support software framework for engineering quality intelligence. DQIP is designed to integrate data ingestion, validation, statistical process control, Six Sigma quality evaluation, rule-based reasoning, visualization, and reporting into a single reusable architecture. Its central design idea is the separation of stable software services from variable domain knowledge. The analytical engine, data services, interface components, and reporting services are domain-independent, whereas quality thresholds, rule sets, terminology, and evaluation criteria are configurable. This design enables adaptation to multiple engineering domains while preserving a common software core.

The objective of this paper is not to propose a new statistical formula or a new quality-management standard. Instead, the contribution lies in the software engineering design of an integrated decision-support framework that can operationalize existing quality methods in a reusable and explainable manner. The paper describes the requirements of engineering process, the layered architecture, the implementation strategy, and the software-oriented evaluation of the DQIP prototype. The intended contribution is therefore positioned at the intersection of software architecture, engineering informatics, decision-support systems, and quality intelligence.

## 2. NOVELTY AND RESEARCH CONTRIBUTIONS

The novelty of DQIP lies in treating engineering quality intelligence as a software architecture problem rather than as a collection of isolated statistical calculations. In many engineering environments, quality evaluation is performed by combining spreadsheets, manual acceptance checks, control charts, project-specific templates, and expert judgment. While each activity may be valid individually, the absence of integration limits reproducibility and makes it difficult to reuse the same workflow across projects. DQIP contributes a modular architecture that transforms these activities into coordinated software services.

The first contribution is a layered decision-support architecture for engineering quality intelligence. The architecture separates presentation, application, analytics, reasoning, data management, and infrastructure responsibilities. This separation reduces coupling between user-interface components, analytical modules, rule definitions, and persistent data. As a result, changes in one layer can be made with limited impact on other layers. This is important for engineering applications because new indicators, input formats, or quality criteria are frequently introduced during project execution.

The second contribution is the integration of statistical quality analysis and explainable reasoning within a single software pipeline. Conventional statistical process control tools usually stop at charts or indices, leaving the interpretation to the user. DQIP links process statistics, Six Sigma indicators, thresholds, and rule-based conditions to generate structured recommendations. The decision output is not only a result value but a traceable explanation showing which data conditions, rules, and analytical indicators contributed to the recommendation.

The third contribution is a configurable domain-knowledge mechanism. Instead of hard-coding quality rules into the program logic, DQIP represents domain thresholds, acceptance criteria, and decision rules as configurable knowledge assets. This allows the same analytical services to be reused for different materials, projects, inspection categories, or engineering sectors. The approach improves adaptability and supports gradual extension as organizational knowledge matures.

The fourth contribution is a reusable engineering analytics pipeline that combines ingestion, schema validation, preprocessing, analysis, visualization, recommendation, and reporting. The pipeline reduces manual movement of data between software tools and supports traceability from raw input to final decision output. This is particularly valuable where quality decisions must be audited or justified to clients, regulators, or internal review teams.

The fifth contribution is the prototype implementation of a lightweight open-source platform. The prototype demonstrates that a practical decision-support environment can be implemented without dependence on proprietary

enterprise platforms. Python-based analytical libraries, interactive dashboard components, and modular service design are combined to provide an accessible implementation pathway for researchers and engineering organizations.

The sixth contribution is the explicit evaluation of the platform using software quality attributes. Rather than evaluating only engineering outcomes, the paper examines functional adequacy, maintainability, modularity, usability, interoperability, performance behavior, and cross-domain reuse. This evaluation perspective aligns the work with software engineering research and distinguishes it from purely domain-specific quality-control studies.

Overall, DQIP contributes a reusable framework for transforming engineering quality data into actionable decision intelligence. The proposed platform can support current quality-management tasks while providing a foundation for future predictive analytics, digital twin integration, and cloud-native quality services.

## 3. RELATED WORK

Decision-support systems have been applied across management, manufacturing, engineering, healthcare, and logistics to support decisions that cannot be fully automated by deterministic computation alone. The classical view of a decision-support system combines data resources, models, and human interaction to improve the quality of semi-structured decisions [11]-[13]. Engineering quality management fits this category because decisions must often combine measured values, tolerance limits, statistical behavior, project constraints, and expert interpretation. A system that only stores data is insufficient; a system that only calculates indicators is also insufficient. The practical requirement is an integrated environment that connects data, models, and explanations.

Quality-management systems and enterprise quality software typically emphasize documentation control, audit trails, non-conformance management, corrective-action workflows, and compliance with standards such as ISO 9001 [15]. These capabilities are essential for organizational quality assurance. However, enterprise systems are often configured around forms and approval processes rather than flexible analytical workflows. When engineers require project-specific statistical evaluation or custom dashboard indicators, additional spreadsheet processing or external analysis tools are commonly used. This creates a separation between compliance management and engineering analytics.

Statistical process control and Six Sigma methods provide well-established techniques for monitoring process stability and evaluating capability [14], [16]-[18]. Control charts, process capability indices, defect metrics, and variation analysis can reveal process drift and support corrective action. Nevertheless, most statistical tools are analytical utilities rather than full decision-support platforms. They provide numerical or graphical evidence, but the reasoning that connects evidence to engineering recommendations is frequently informal. This gap becomes critical when organizations need repeatable and auditable decision logic.

Dashboard and business-intelligence platforms have improved the ability of users to explore data visually [19], [20]. Interactive charts, filters, and summarized indicators are valuable for situational awareness. Yet dashboards alone do not guarantee decision quality. Without validated data pipelines, domain-specific rules, and traceable interpretation mechanisms, dashboards may simply accelerate the display of inconsistent or incomplete data. Engineering quality intelligence requires visualization to be connected to data validation and reasoning rather than treated as a separate reporting layer.

The growth of Industry 4.0, smart manufacturing, and digital twin research has further increased interest in integrated engineering information systems [21]-[24]. These approaches emphasize connectivity, cyber-physical integration, simulation, and feedback loops. However, large-scale digital twin environments can be complex and costly to implement. Many small and medium engineering organizations require an intermediate solution: a modular, lightweight, and extensible software platform that can improve quality intelligence without requiring a complete digital twin infrastructure. DQIP addresses this need by providing an adaptable decision-support framework that can later be connected to larger digital ecosystems.

Recent advances in data analytics and machine learning have also influenced engineering decision support [25]-[28]. Predictive models can detect anomalies, forecast failures, and classify quality risks. However, data-driven models introduce issues of model governance, explainability, data drift, and technical debt [27]. In quality-management contexts, users often need to understand why a recommendation was made. Therefore, rule-based and statistical reasoning remain important, especially when organizational standards, acceptance limits, and regulatory requirements must be explicitly represented. DQIP is designed to support current statistical and rule-based reasoning while allowing future predictive services to be integrated as additional modules.

From a software architecture perspective, the literature emphasizes modularity, service orientation, data-intensive design, and separation of concerns [7]-[10], [35]-[38]. These principles are directly relevant to engineering analytics platforms because input schemas, analytical functions, reporting formats, and decision rules evolve over time. A tightly coupled application may work initially but becomes difficult to maintain as new use cases appear. DQIP applies these principles by separating analytical services from decision rules and user-interface components.

Table 1 summarizes the main limitations observed in existing software categories and identifies how DQIP addresses each limitation through architectural design. The comparison shows that DQIP is not intended to replace every specialized tool. Instead, it provides an integration-oriented framework that combines the essential capabilities needed for engineering quality intelligence

**Table 1:** Comparative review of existing software categories and DQIP

| Software category | Typical capability | Observed limitation | DQIP response |
|---|---|---|---|
| Enterprise quality-management systems | Document control, audits, non-conformance workflow | Limited custom analytics and project-specific reasoning | Adds statistical services and configurable decision logic |
| SPC and Six Sigma tools | Charts, capability indices, defect metrics | Outputs require separate expert interpretation and reporting | Connects indicators to explainable rules and recommendations |
| Dashboard/BI platforms | Interactive visualization and filters | Dependent on external data preparation and manual reasoning | Integrates validation, analytics, reasoning, and reports |
| Spreadsheet workflows | Flexible ad hoc calculations | Weak traceability, formula risk, poor reuse | Provides reusable modules and auditable processing |
| AI/ML quality applications | Prediction, anomaly detection, classification | May be domain-specific, opaque, and maintenance-heavy | Allows AI services to be added within governed architecture |

## 4. REQUIREMENTS ENGINEERING AND RESEARCH METHODOLOGY

The development of DQIP followed a requirements-driven software engineering process. The objective was to identify the functions and quality attributes required for a reusable engineering decision-support platform. The requirements analysis considered the needs of quality engineers, project managers, inspection teams, data analysts, software maintainers, and organizational decision-makers. These stakeholders interact with quality data at different levels of abstraction. Inspectors generate and validate raw records. Quality engineers evaluate conformance and variation. Managers require summarized indicators and risk information. Software maintainers require modularity and testability. The platform must therefore support both technical analysis and organizational decision workflows.

The first step of the methodology was problem decomposition. Engineering quality workflows were decomposed into data acquisition, data validation, preprocessing, statistical evaluation, rule interpretation, visualization, reporting, and decision recording. This decomposition made it possible to identify reusable software services rather than design a monolithic application. Each workflow activity was mapped to a candidate module with clear input, processing, and output responsibilities.

The second step was the requirements classification. Functional requirements describe what the platform must do, such as importing datasets, validating schemas, computing quality indicators, generating dashboards, applying decision rules, and exporting reports. Non-functional requirements describe how the platform must behave, including maintainability, usability, interoperability, reliability, responsiveness, security, portability, and extensibility. The distinction is important because an engineering analytics tool may satisfy basic functional requirements while still failing as a long-term software system if it is difficult to maintain or adapt.

The third step was architectural design. Based on the identified requirements, layered architecture was selected to separate responsibilities and reduce coupling. The presentation layer handles user interaction and visualization. The application layer coordinates workflows and session logic. The analytics layer executes statistical and quality functions. The reasoning layer evaluates configurable rules and generates recommendations. The data-management layer handles input, validation, storage, and retrieval. The infrastructure layer supports deployment, logging, configuration, and execution services. The resulting architecture is shown conceptually in Figure 3.

The fourth step was prototype implementation. A lightweight prototype was developed using Python-based open-source technologies. The implementation focused on demonstrating the feasibility of the architecture rather than delivering a complete enterprise product. The prototype includes dataset upload, preprocessing, descriptive statistics, quality metrics, control-chart generation, rule-based interpretation, dashboard display, and report export. The prototype was designed so that analytical modules and rule sets could be extended without changing the entire application.

The fifth step was software-oriented evaluation. The evaluation considered whether the platform satisfied the identified requirements and whether the modular architecture supported maintainability and cross-domain reuse. Evaluation criteria were derived from software quality models and software engineering standards [3]-[6]. The assessment focused on functional adequacy, performance behavior, usability, compatibility, maintainability, portability, and extensibility. The goal was not to claim universal industrial validation, but to verify that the proposed architecture provides a practical and coherent foundation for decision-support software in engineering quality management.

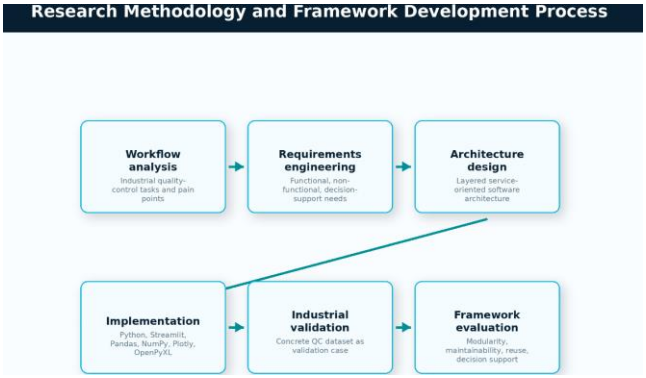


**Figure 1:** Overall research methodology used for developing the Digital Quality Intelligence Platform.

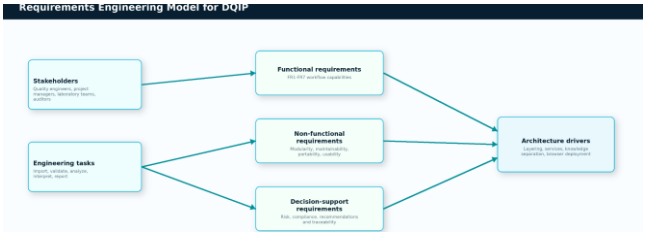


**Figure 2:** Requirements engineering workflow for converting stakeholder needs into platform services.

Figure 1 presents the overall research methodology. Figure 2 presents the requirements for engineering workflow used to translate stakeholder needs into architectural components. The methodology emphasizes iterative refinement because engineering software requirements often evolve as users interact with dashboards and discover new analytical needs.

Table 2 summarizes the functional and non-functional requirements identified during the requirements engineering phase and the corresponding software mechanisms used in DQIP.

**Table 2:** Functional and non-functional requirements matrix

| Req. ID | Requirement | Software concern | DQIP mechanism |
|---|---|---|---|
| FR1 | Import heterogeneous quality datasets | Functional adequacy | File ingestion, schema mapping, canonical model |
| FR2 | Validate input quality | Reliability | Missing-value, type, range, duplicate, and schema checks |
| FR3 | Compute statistical indicators | Analytical correctness | Reusable analytics services for SPC and quality metrics |
| FR4 | Generate recommendations | Decision support | Configurable rule engine and explanation generator |
| FR5 | Visualize results | Usability | Interactive dashboard, charts, filters, and status cards |
| FR6 | Export decision reports | Traceability | Report service using shared analytical outputs |
| NFR1 | Support maintainability | Maintainability | Layered architecture and low coupling |
| NFR2 | Support cross-domain reuse | Portability/extensibility | Domain profiles, configurable thresholds, reusable services |

## 5. SOFTWARE ARCHITECTURE

The software architecture of DQIP is the central contribution of this work. The platform is designed as a modular decision-support framework in which each major concern is encapsulated in a layer or service. This structure improves maintainability, reduces the risk of uncontrolled dependencies, and enables domain-specific configuration. The architecture is influenced by established software architecture principles, including separation of concerns, service modularity, layered organization, and data-intensive application design [7]-[10].

The presentation layer provides a user-facing interface. It includes dashboards, data upload controls, filters, charts, summary cards, recommendation views, and report-download options. The layer is intentionally kept separate from analytical logic. A dashboard component requests processed results from application services rather than directly manipulating statistical functions. This separation allows the user interface to be redesigned or replaced without changing the analytical engine. It also supports future deployment through web, desktop, or enterprise portal interfaces.

The application layer coordinates the workflow of the platform. It receives user events, invokes data services, triggers analytical modules, calls the reasoning engine, and prepares outputs for visualization. This layer acts as the orchestration mechanism of DQIP. For example, when a user uploads a quality dataset, the application layer initiates schema validation, preprocessing, statistical calculation, rule evaluation, visualization assembly, and report generation. By centralizing orchestration, the platform avoids distributing workflow logic across unrelated modules.

The analytics layer contains reusable computational services. These services include descriptive statistics, trend analysis, statistical process control, capability analysis, Six Sigma indicators, defect-rate evaluation, and comparative summaries. Each analytical service exposes a defined interface and returns structured results. This design allows new analytical functions to be added without rewriting the dashboard or reasoning engine. For example, an anomaly-detection service or predictive model can later be added as an independent analytics module that consumes validated data and returns a standardized output object.

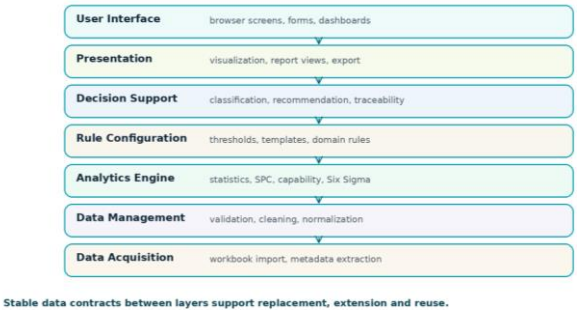


**Figure 3:** Layered software architecture of DQIP

The reasoning layer transforms analytical evidence into decision-support output. It contains the rule engine, knowledge repository, threshold definitions, recommendation templates, and explanation generator. Unlike hard-coded conditional logic scattered throughout the application, DQIP stores domain rules as configurable knowledge elements. A rule may specify that a process should be flagged if a capability index falls below a threshold, if control-chart violations are detected, or if defect frequency exceeds an acceptable range. The reasoning layer evaluates these conditions and produces recommendations with traceable explanations. This is especially important in engineering contexts where users must justify corrective actions.

The data-management layer handles data ingestion, schema validation, cleaning, transformation, persistence, and retrieval. Engineering datasets are often heterogeneous because different projects use different column names, units, inspection categories, and reporting conventions. The data-management layer therefore includes mapping and validation mechanisms that convert raw records into a canonical internal structure. Data validation checks missing values, type inconsistencies, range errors, duplicate records, and schema violations before analytical processing begins. This reduces the risk of misleading results caused by poor data quality.

The infrastructure layer provides configuration, logging, deployment support, security boundaries, file management,

and integration hooks. In the prototype, this layer is lightweight, but it is designed to support future enterprise deployment. Logging is essential for auditability because quality decisions may need to be traced after a project review. Configuration support allows different domain profiles to be loaded without changing program code. Integration hooks provide a path for connecting DQIP to databases, laboratory information systems, enterprise quality-management platforms, and digital twin environments.

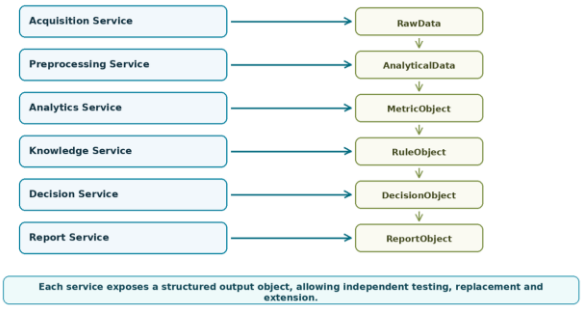


**Figure 4:** Component architecture and service interaction model.

The component architecture shown in Figure 4 illustrates how the services interact. The user interface communicates with the application controller. The controller interacts with the data service to obtain validated datasets. Analytical services compute indicators and return structured metrics. The reasoning engine evaluates rules using analytical outputs and configuration data. The reporting service assembles charts, tables, explanations, and recommendations into exportable formats. This structure creates a clear flow from data to decision while preserving modular independence.

A key architectural feature of DQIP is the use of a canonical data model. Raw input datasets may vary across domains, but analytical services require consistent data structures. The canonical model includes common concepts such as observation identifier, timestamp, parameter name, measured value, specification limit, tolerance band, source, inspection category, project identifier, and status. Domain-specific attributes can be added through metadata fields. This model enables statistical services to operate on standardized inputs while allowing flexible domain extension.

Another important feature is the configuration-driven rule model. Domain rules are represented separately from the execution engine. A rule definition includes a condition, threshold, severity level, explanation text, and recommended action. For example, a rule may classify a process as needing investigation if statistical stability is violated and defect frequency is increasing. The engine evaluates the condition using analytical outputs and returns the appropriate recommendation. Because rule definitions are configurable, users can adjust thresholds for different projects or standards without modifying source code.

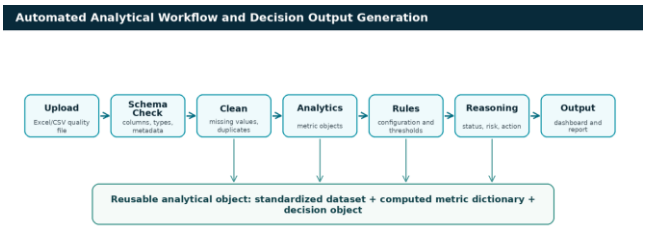


**Figure 5:** Decision-support pipeline from raw engineering data to explainable recommendation.

The decision-support pipeline shown in Figure 5 combines the major architectural functions into an end-to-end flow. First, data are acquired from user uploads or external systems. Second, validation and preprocessing convert the data into the canonical model. Third, analytical modules calculate quality indicators. Fourth, the reasoning engine evaluates rules against the calculated indicators. Fifth, dashboards and reports communicate the results to users. Finally, feedback from user decisions can be stored for future knowledge refinement. This pipeline supports both immediate analysis and organizational learning.

The architecture also supports incremental extensibility. New visualization components can be added to the presentation layer. New workflow actions can be added in the application layer. New statistical models can be added to the analytics layer. New domain rules can be added to the reasoning layer. New storage mechanisms can be added to the data-management layer. New deployment strategies can be added to the infrastructure layer. This extensibility is essential because engineering organizations rarely adopt a complete decision-support ecosystem at once. They usually begin with a narrow use case and expand as benefits are demonstrated.

From a maintainability perspective, modular architecture reduces the probability that a change in one function will break unrelated functions. For example, if a new process capability index is added, the change is localized within an analytical service and visualization mapping. If a new acceptance rule is required, the change is localized within the knowledge configuration. If the dashboard layout changes, the analytical engine remains unaffected. This separation supports unit testing, regression testing, and future refactoring.

From an interoperability perspective, DQIP can exchange data with external systems through standardized import and export formats. The prototype supports tabular file-based input, but the architecture can be extended to database connections, REST interfaces, or message queues. The use of structured outputs from analytical and reasoning services also enables downstream integration with enterprise reporting systems.

The architecture deliberately balances simplicity and extensibility. A fully enterprise-grade microservices system would increase deployment complexity for early adoption. A monolithic script would be easier to write but difficult to maintain. DQIP adopts a modular layered structure that can run as a lightweight application while preserving a path toward distributed deployment. This makes the framework

suitable for research prototypes, small engineering teams, and future enterprise scaling.

## 6. IMPLEMENTATION

The DQIP prototype was implemented using open-source software components to demonstrate the feasibility of the proposed architecture. Python was selected as the primary implementation language because of its mature ecosystem for data processing, statistical analysis, visualization, and rapid prototyping. Libraries such as Pandas and NumPy support data manipulation and numerical computation [29], [30]. SciPy and related scientific libraries provide extensibility for advanced statistical functions [31]. Visualization components are implemented using Matplotlib and interactive dashboard libraries such as Plotly [32], [33]. For future large-scale deployment and distributed data analytics, the architecture can also be extended using big-data processing frameworks such as Apache Spark [34].

The implementation follows a modular package structure. The data module handles input loading, schema mapping, data cleaning, type conversion, and validation. The analytics module contains functions for descriptive statistics, trend indicators, control-chart preparation, process capability analysis, defect-rate summaries, and Six Sigma-related calculations. The reasoning module contains rule definitions, threshold evaluation, severity classification, explanation generation, and recommendation assembly. The visualization module prepares charts, tables, summary cards, and dashboard layouts. The reporting module exports analytical summaries and recommendations in reusable formats.

The user workflow begins with dataset upload. The uploaded dataset is first checked for required fields and data-type consistency. Missing or invalid values are identified and reported to the user. Where appropriate, the system applies preprocessing rules such as numeric conversion, unit normalization, date parsing, and duplicate removal. The cleaned dataset is then transformed into the internal canonical model used by the analytics layer. This conversion is important because it isolates analytical services from the variability of external file formats.

After preprocessing, the analytics module calculates quality indicators. Depending on the dataset and selected domain profile, the prototype can generate descriptive summaries, variation measures, process stability indicators, capability-related measures, defect distributions, and time-based trends. The outputs are returned as structured dictionaries or data frames that can be consumed by the reasoning and visualization layers. Analytical functions are designed to be independent so that they can be tested separately and reused in different workflows.

The reasoning module applies configurable rules to analytical outputs. A rule definition includes a logical condition, threshold values, severity level, message template, and recommendation text. For example, if the defect rate exceeds a configured threshold, the system may classify the issue as moderate or severe and recommend root-cause investigation. If the process variation is high, the system may recommend process stabilization before acceptance decisions are made. The reasoning output includes the triggering condition and the evidence values used by the rule, thereby improving transparency.

The dashboard interface presents the results through summary indicators, charts, tables, alerts, and recommendations. Figure 6 shows the dashboard placeholder for the final manuscript. The dashboard is designed to support rapid interpretation. Summary cards provide high-level status. Charts display variation, defect concentration, and trend behavior. Recommendation panels explain the decision logic. Filters allow users to focus on specific projects, inspection categories, parameters, or time periods.

Report generation is implemented as a separate service so that the dashboard and exported report use the same analytical results. This avoids inconsistency between what users see interactively and what is documented for review. Reports can include dataset information, validation messages, statistical summaries, charts, rule-trigger explanations, and recommended actions. This feature supports auditability and knowledge reuse.

Deployment flexibility is a design goal of the prototype. The application can run locally for research use, be hosted through a lightweight web service for team access or be packaged for cloud deployment. The platform can also be published as a demonstrator on open software hosting environments so that users can interact with the prototype without installing the full development stack. The prototype architecture therefore supports reproducibility and future collaborative extension.

The implementation is intentionally extensible rather than overly specialized. New engineering domains can be added by defining new schema mappings, thresholds, terminology, and rule profiles. New statistical functions can be added as analytics services. New reports can be added through the reporting layer. This modular implementation confirms the practicality of the layered architecture and provides a basis for future research on AI-assisted engineering quality intelligence.


**Figure 6:** Prototype dashboard interface for engineering quality intelligence.

## 7. RESULTS AND DISCUSSION

The evaluation of DQIP was conducted from a software engineering perspective. The objective was to determine whether the platform satisfies the architectural and functional goals identified during requirements engineering. The evaluation considered functional adequacy, modularity, maintainability, usability, interoperability, responsiveness, traceability, and cross-domain adaptability. These attributes are consistent with software quality models and software engineering standards [3]-[6].

Functional adequacy was evaluated by checking whether the prototype supports the complete data-to-decision workflow. The platform accepts quality datasets, validates schema requirements, preprocesses records, calculates statistical indicators, applies decision rules, presents dashboards, and exports reports. This confirms that DQIP operates as an integrated decision-support environment rather than a standalone statistical script. The most important functional benefit is the continuity of the workflow: raw input records can be traced through validation, analysis, interpretation, and reporting.

Modularity was evaluated by examining whether major functions are separated into independent services. The prototype separates data handling, analytics, reasoning, visualization, and reporting. This separation allows individual components to be modified or extended with limited impact on the rest of the platform. For example, a new chart type can be added to the visualization module without changing the reasoning engine. Similarly, a new acceptance rule can be added to the knowledge configuration without rewriting statistical functions.

Maintainability was evaluated through change scenarios. Three representative changes were considered: adding a new quality indicator, modifying an acceptance threshold, and changing the dashboard layout. In a tightly coupled spreadsheet workflow, each change may require manual formula edits, chart changes, and repeated verification across multiple files. In DQIP, the changes are localized to the corresponding module or configuration item. This improves maintainability and reduces the risk of inconsistent updates.

Usability was evaluated by considering the clarity of user interaction and decision output. The dashboard design emphasizes summary indicators, visual trends, alerts, and explanation panels. Users are not required to inspect raw calculations to understand why a recommendation was generated. Instead, the recommendation includes the triggering rule and relevant analytical evidence. This supports explainable decision-making and reduces dependence on implicit expert interpretation.

Interoperability was evaluated through the ability of the platform to accept common tabular engineering datasets and transform them into a canonical internal format. The prototype uses file-based input, which is appropriate for early deployment and research validation. However, the architecture is not limited to file input. The same data-management layer can be extended to connect with relational databases, laboratory information systems, enterprise quality platforms, or sensor data services. This confirms that the design is compatible with future integration requirements.

Performance behavior was evaluated qualitatively by examining the responsiveness of the prototype during typical analytical operations. The modular Python implementation supports interactive use for representative engineering datasets and provides dashboard updates without requiring users to perform separate spreadsheet calculations. For large enterprise datasets, future scaling can be achieved by optimizing data-processing routines, introducing database-backed storage, or distributing analytics services. The present evaluation confirms feasibility at the prototype level while identifying scalability as a future enhancement area.

Traceability is one of the most important benefits of the platform. Traditional spreadsheet workflows often hide the connection between input records, formulas, charts, and decisions. DQIP makes this connection explicit. Validation logs show whether data were accepted or corrected. Analytical outputs show calculated indicators. Rule explanations show which conditions triggered recommendations. Reports preserve the decision context. This traceability is important for engineering audits, corrective-action reviews, and organizational learning.

The results also show that DQIP reduces workflow fragmentation. In conventional practice, engineers may use one tool for data cleaning, another for statistical calculation, another for charting, and another for reporting. DQIP integrates these functions into a single architecture. This does not eliminate the need for expert judgment, but it improves the consistency and transparency of the information presented to experts. Decision-makers can focus on interpretation and corrective action rather than manual data movement.

Table 3 summarizes the software evaluation results. The table reports the evaluation focus, evidence considered in the prototype, and interpretation. The findings indicate that DQIP meets the primary objective of providing a modular and reusable software framework for engineering quality intelligence. The evaluation also identifies areas for future work, including stronger security controls, enterprise authentication, automated test suites, database-backed persistence, and cloud-native deployment.

The discussion highlights an important distinction between analytical accuracy and software quality. Statistical formulas such as control-chart limits or capability indices are well established. The challenge addressed by DQIP is not mathematical novelty but software integration. By embedding established analytical methods within a modular and explainable decision-support framework, DQIP improves the practical usability of engineering

quality data. This software engineering contribution is significant because many quality failures arise not from a lack of data but from delayed, inconsistent, or poorly traceable interpretation of data.

The prototype results also suggest that lightweight open-source platforms can provide meaningful quality-intelligence capabilities before organizations invest in large enterprise systems. This is valuable for small and medium organizations that need better analytics but cannot immediately deploy a complex digital twin or enterprise-quality infrastructure. DQIP can serve as a bridge between spreadsheet-based practice and more advanced intelligent quality-management systems.

A further implication is that quality logic becomes a maintainable software asset when it is encoded as a governed rule profile instead of being distributed across spreadsheets and individual reports. In conventional workflows, the same acceptance criterion may be implemented differently by different engineers, especially when templates are copied and modified over several projects. DQIP reduces this risk by centralizing rule definitions and using the same rule engine for dashboard alerts and report generation. This improves consistency and provides a clearer basis for review when a threshold or recommendation must be changed.

The evaluation also identifies limitations that must be considered before large-scale adoption. The prototype demonstrates architectural feasibility and functional integration, but it does not replace project-specific validation by domain experts. Each domain profile must be reviewed to ensure that thresholds, units, and severity classes match the governing specification. The current prototype also emphasizes file-based input and lightweight deployment; enterprise environments will require authentication, role-based authorization, database persistence, backup strategy, and controlled change management. These limitations do not invalidate the proposed framework, but they define the next engineering steps needed to transform the prototype into a production-grade platform.

From a research perspective, the result shows that decision-support value can be created without relying exclusively on black-box prediction. Transparent statistical indicators and configurable rules remain useful where engineering decisions require justification. Machine learning can later be introduced as an additional analytics service, but the rule and explanation layers provide an essential governance mechanism for accepting, rejecting, or contextualizing model outputs. This makes DQIP compatible with future AI-enabled quality intelligence while preserving explainability.

**Table 3:** Software evaluation summary

| Evaluation attribute | Evaluation basis | Prototype observation | Interpretation |
|---|---|---|---|
| Functional adequacy | End-to-end workflow execution | Import, validation, analytics, reasoning, dashboard, and reporting supported | Primary decision-support workflow is complete |
| Modularity | Component separation | Data, analytics, reasoning, visualization, and reporting isolated | Supports independent maintenance and extension |
| Maintainability | Representative change scenarios | New rules and indicators localized to configuration or service modules | Reduces spreadsheet-style change risk |
| Usability | Dashboard clarity and recommendation explanation | Summary indicators, charts, alerts, and rule explanations presented | Improves interpretability for engineering users |
| Interoperability | Common tabular input and canonical model | File-based ingestion converted to internal schema | Provides path for future database/API integration |
| Traceability | Input-to-output audit path | Validation messages, metrics, triggered rules, and reports preserved | Improves auditability of quality decisions |
| Extensibility | New domain and analytics scenarios | Domain profiles and modular services can be extended | Supports reuse beyond one quality use case |

## 8. CROSS-DOMAIN ADAPTABILITY

Cross-domain adaptability is a central design objective of DQIP. Engineering quality problems differ across domains, but many software functions are reusable. A manufacturing process, a construction-material testing workflow, a maintenance inspection program, and a laboratory quality-control system may use different terminology and thresholds, yet all require data validation, statistical summary, variation monitoring, rule interpretation, visualization, and reporting. DQIP exploits this common structure by separating reusable platform services from configurable domain knowledge.

Figure 7 presents the cross-domain adaptability model. At the center of the model is the DQIP core, consisting of data services, analytics services, reasoning services, visualization services, and reporting services. Around the core are domain profiles. A profile defines input schema mappings, parameter names, acceptance limits, rule sets, units, severity classes, and report terminology. When a new domain is introduced, the platform does not require redevelopment of the core engine. Instead, the relevant domain profile is configured and linked to the existing services.

This design is particularly useful for organizations that operate across multiple engineering activities. For example, an infrastructure organization may need to evaluate concrete test results, welding inspection records, pavement quality indicators, and equipment maintenance logs. Although these datasets differ, the decision-support workflow is structurally similar. DQIP can provide a common software environment while allowing each domain to retain its own rules and terminology.

The adaptability model also supports gradual maturity. An organization may begin with simple descriptive statistics and threshold rules. Later, it may add control-chart logic,

capability analysis, anomaly detection, predictive models, or digital twin integration. Because DQIP is modular, these enhancements can be introduced incrementally. This avoids the risk of replacing the entire platform whenever analytical maturity improves.

Another benefit is knowledge standardization. When domain rules are represented explicitly, organizational knowledge becomes reusable rather than remaining hidden in spreadsheets or individual expert practices. New engineers can learn from the rule explanations and reports. Managers can compare quality indicators across projects. Software maintainers can update configurations systematically. This supports both technical consistency and organizational learning.

Cross-domain reuse does not mean that domain expertise is unnecessary. Each domain still requires experts to define appropriate thresholds, interpret results, and validate recommendations. DQIP provides the software structure through which domain expertise can be represented and applied consistently. The platform therefore complements expert judgment rather than replacing it.

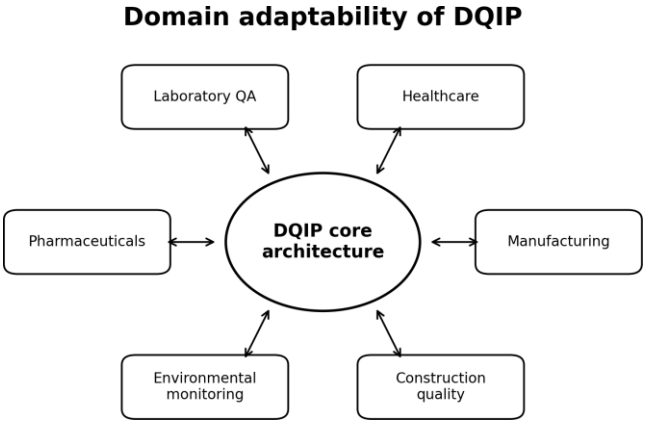


**Figure 7:** Cross-domain adaptability model for reusing DQIP services across engineering quality domains.

## 9. CONCLUSION

This paper presented the Digital Quality Intelligence Platform, a modular decision-support software framework for engineering quality intelligence. The work addressed a common problem in engineering organizations: quality data is abundant, but the software workflows used to transform data into decisions are often fragmented, manual, and difficult to reuse. DQIP responds to this problem by integrating data ingestion, validation, statistical quality evaluation, rule-based reasoning, visualization, reporting, and recommendation generation within a layered software architecture.

The principal contribution of the research is architectural. DQIP separates presentation, application, analytics, reasoning, data management, and infrastructure responsibilities so that each part of the system can evolve independently. The platform also separates domain-independent software services from configurable domain knowledge, enabling cross-domain reuse without redesigning the core engine. The prototype implementation demonstrates the feasibility of the architecture using open-source Python-based technologies and interactive dashboard components.

The software-oriented evaluation indicates that DQIP improves workflow integration, traceability, maintainability, explainability, and adaptability compared with disconnected spreadsheet-based approaches. The platform is not intended to eliminate engineering expertise. Instead, it provides a structured computational environment that helps experts interpret quality data consistently and document decisions transparently.

Future work will extend DQIP in five directions. First, predictive analytics and machine learning services can be integrated as optional modules. Second, database-backed persistence and enterprise authentication can be added for organizational deployment. Third, automated testing and continuous integration pipelines can strengthen software reliability. Fourth, cloud-native and microservice deployment patterns can support larger datasets and multi-user access. Fifth, integration with digital twin environments can enable real-time quality intelligence. These extensions will further develop DQIP as a reusable software engineering framework for intelligent engineering decision support.

## ACKNOWLEDGEMENT

The author thanks St. Joseph's College of Engineering (Autonomous), Chennai, India, for academic support and research encouragement. The author also acknowledge the operational engineering quality-control records used for validating the software workflow. The records were used only to verify data ingestion, preprocessing, analytics, dashboard generation and decision-support functions, and no project, organization or individual identity is disclosed.

## FUNDING

This research did not receive any specific grant from funding agencies in the public, commercial or non-profit sectors.

## CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

## AVAILABILITY OF DATA AND MATERIALS

The validation data used in this study was derived from anonymized engineering quality-control records. The data are available from the corresponding author upon reasonable request, subject to institutional permission and confidentiality restrictions. The dataset is used in the manuscript only as a validation case for verifying the software workflow and not as a domain-specific engineering claim.

## SOFTWARE AVAILABILITY

The Digital Quality Intelligence Platform (DQIP) prototype is available online for demonstration and evaluation purposes through the Hugging Face Spaces deployment environment. The software may be accessed at:

https://huggingface.co/spaces/janicecodes/Concrete_strength_analysis_and_6_sigma_intelligence.

The deployed prototype demonstrates the browser-based workflow for workbook upload, data validation, analytical processing, rule-based interpretation, dashboard visualization and report-oriented decision support.

## REFERENCES


[1] R. S. Pressman and B. R. Maxim. **Software Engineering: A Practitioner's Approach,** *9th ed. New York, NY: McGraw-Hill, 2020.*

[2] I. Sommerville. **Software Engineering**, *10th ed. Boston, MA: Pearson, 2016.*

[3] *ISO/IEC/IEEE 42010:2011.* **Systems and Software Engineering—Architecture Description.** *Geneva, Switzerland: ISO, 2011.*

[4] *ISO/IEC/IEEE 29148:2018.* **Systems and Software Engineering—Life Cycle Processes—Requirements Engineering.** *Geneva, Switzerland: ISO, 2018.*

[5] *ISO/IEC 25010:2011.* **Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models.** *Geneva, Switzerland: ISO, 2011.*

[6] *IEEE Std 730-2014. IEEE* **Standard for Software Quality Assurance Processes**. *New York, NY: IEEE, 2014.*

[7] L. Bass, P. Clements, and R. Kazman. **Software Architecture in Practice**, *4th ed. Boston, MA: Addison-Wesley, 2021.*

[8] M. Fowler. **Patterns of Enterprise Application Architecture**. *Boston, MA: Addison-Wesley, 2002.*

[9] S. Newman. **Building Microservices,** *2nd ed. Sebastopol, CA: O'Reilly Media, 2021.*

[10] M. Kleppmann. **Designing Data-Intensive Applications.** *Sebastopol, CA: O'Reilly Media, 2017.*

[11] D. J. Power. Decision **Support Systems: Concepts and Resources for Managers.** *Westport, CT: Quorum Books, 2002.*

[12] E. Turban, J. E. Aronson, T.-P. Liang, and R. Sharda. **Decision Support and Business Intelligence Systems,** *8th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007.*

[13] H. A. Simon. **The New Science of Management Decision.** *New York, NY: Harper & Row, 1960*.

[14] D. C. Montgomery**. Introduction to Statistical Quality Control,** *8th ed. Hoboken, NJ: Wiley, 2019.*

[15] ISO 9001:2015. **Quality Management Systems—Requirements.** Geneva, Switzerland: ISO, 2015.

[16] *ISO 7870-2:2013*. **Control Charts—Part 2: Shewhart Control Charts**. *Geneva, Switzerland: ISO, 2013.*

[17] T. Pyzdek and P. Keller. **The Six Sigma Handbook,** *5th ed. New York, NY: McGraw-Hill Education, 2018*.

[18] W. A. Shewhart. **Economic Control of Quality of Manufactured Product**. *New York, NY: D. Van Nostrand Company, 1931.*

[19] T. H. Davenport and J. G. Harris. **Competing on Analytics: The New Science of Winning**. *Boston, MA: Harvard Business School Press, 2007.*

[20] A. Gandomi and M. Haider. **Beyond the hype: Big data concepts, methods, and analytics,** *International Journal of Information Management, vol. 35, no. 2, pp. 137-144, 2015.*

[21] Y. Lu. Industry 4.0: **A survey on technologies, applications and open research issues**, *Journal of Industrial Information Integration, vol. 6, pp. 1-10, 2017.*

[22] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee. **Digital twin in industry: State-of-the-art,** *IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2405-2415, 2019.*

[23] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn. **Digital Twin in manufacturing: A categorical literature review and classification,** *IFAC-PapersOnLine, vol. 51, no. 11, pp. 1016-1022, 2018.*

[24] H. Kagermann, W. Wahlster, and J. Helbig. **Recommendations for Implementing the Strategic Initiative Industrie 4.0**. *Frankfurt, Germany: acatech, 2013.*

[25] S. F. Wamba, A. Gunasekaran, S. Akter, S. J. Ren, R. Dubey, and S. J. Childe. **Big data analytics and firm performance: Effects of dynamic capabilities**, *Journal of Business Research, vol. 70, pp. 356-365, 2017.*

[26] A. Janiesch, P. Zschech, and K. Heinrich. **Machine learning and deep learning,** *Electronic Markets, vol. 31, pp. 685-695, 2021.*

[27] D. Sculley et al. **Hidden technical debt in machine learning systems, in Proc. Advances in Neural Information Processing Systems**, *2015, pp. 2503-2511.*

[28] L. Breiman**. Statistical modeling: The two cultures, Statistical Science,** *vol. 16, no. 3, pp. 199-231, 2001.*

[29] W. McKinney. **Data structures for statistical computing in Python,** *in Proc. 9th Python in Science Conference, 2010, pp. 56-61.*

[30] C. R. Harris et al**. Array programming with NumPy, Nature,** *vol. 585, pp. 357-362, 2020.*

[31] P. Virtanen et al. SciPy 1.0: **Fundamental algorithms for scientific computing in Python, Nature Methods**, *vol. 17, pp. 261-272, 2020.*

[32] J. D. Hunter. Matplotlib: **A 2D graphics environment, Computing in Science & Engineering**, *vol. 9, no. 3, pp. 90-95, 2007.*

[33] *Plotly Technologies Inc*. **Collaborative Data Science.** *Montreal, QC, Canada: Plotly Technologies Inc., 2015*.

[34] M. Zaharia et al. **Apache Spark: A unified engine for big data processing, Communications of the ACM,** *vol. 59, no. 11, pp. 56-65, 2016.*

[35] J. Humble and D. Farley. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.** *Boston, MA: Addison-Wesley, 2010.*

[36] N. Forsgren, J. Humble, and G. Kim. **Accelerate: The Science of Lean Software and DevOps**. *Portland, OR: IT Revolution Press, 2018.*

[37] G. Hohpe and B. Woolf. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.** *Boston, MA: Addison-Wesley, 2003.*

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. **Design Patterns: Elements of Reusable Object-Oriented Software**. *Boston, MA: Addison-Wesley, 1994.*