Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Component-Based Multi-Tier Web Architecture for Enhancing ERP System Development Efficiency and Maintainability

Yonghua Li
Yan Zhang
AOli Jiao
Xinjiang Normal University
Hongle Du
changqiang jing
jingchangqiang@lyu.eud.cn

Linyi University https://orcid.org/0000-0003-3480-5029

Research Article

Keywords: Enterprise Resource Planning, Component-Based Software Engineering, Multi-Tier Architecture, Event-Driven Programming, Software Reuse

Posted Date: August 14th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10313407/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

A Component-Based Multi-Tier Web Architecture for Enhancing ERP System Development Efficiency and Maintainability

Yonghua Li[1], Yan Zhang[1], Lijiao Ao[2], Hongle Du[3], Changqiang Jing[4*]

[1]Engineering Research Center of Qinling Health Welfare Big Data, Shangluo University, Shaanxi , Shangluo，726000, China; ahao050323@163.com(Y.L.);Zhyan@slxy.edu.cn(Y.Z.);
[2]School of Mathematical Sciences, Xinjiang Normal University, Urumqi, 830017; 1576960644@qq.com;
[3]School of Mathematics and Computer Application, Shangluo University, Shaanxi, Shangluo , 726000; dhl5597@163.com;
[4]School of Information Science and Engineering, Linyi University, Linyi 276000, China; jingchangqiang@lyu.edu.cn;
*Correspondence: jingchangqiang@lyu.edu.cn

Abstract: Enterprise Resource Planning (ERP) systems must continuously adapt to rapidly changing business environments while maintaining high levels of scalability, maintainability, and operational reliability. Traditional ERP development approaches often tightly couple user interface logic and business logic, resulting in limited software reusability, increased maintenance complexity, and prolonged development cycles. Furthermore, conventional client–server architectures may experience performance degradation due to excessive database connections and centralized resource utilization. This study proposes a component-based multi-tier web architecture that integrates object-oriented design principles, event-driven programming mechanisms, and a three-tier architecture for ERP system development. The proposed framework encapsulates reusable business functionalities into standardized software components that provide predefined properties, methods, and event interfaces. The architecture separates presentation, business, and data access layers, thereby facilitating collaborative development, improving software maintainability, and enhancing system scalability. To evaluate the effectiveness of the proposed framework, an ERP order-management module was developed. Development productivity, maintainability, deployment efficiency, and system scalability were assessed through a case study conducted in an enterprise software consulting company. The results indicate that the proposed framework reduced development effort by approximately 44% compared with traditional development approaches while significantly improving software reuse and maintenance efficiency. The proposed architecture provides a practical and scalable solution for enterprise information system development and offers valuable guidance for future ERP modernization initiatives.
Keywords: Enterprise Resource Planning, Component-Based Software Engineering, Multi-Tier Architecture, Event-Driven Programming, Software Reuse.

1. Introduction

Enterprise Resource Planning (ERP) systems have become a fundamental component of modern enterprise information infrastructures. By integrating core business functions such as finance, manufacturing, procurement, sales, inventory management, and customer relationship management into a unified platform, ERP systems enable organizations to improve operational efficiency, enhance decision-making capabilities, and maintain competitive advantages in rapidly changing business environments. As enterprises continue to expand their digital transformation initiatives, the demand for scalable, maintainable, and

flexible ERP systems has increased significantly [1, 2].

Traditional ERP systems were commonly developed using client–server architectures in which presentation logic, business logic, and data access functions were tightly coupled. Although such architectures were effective in early enterprise computing environments, they often suffer from several limitations when applied to large-scale and continuously evolving business systems. First, modifications to business requirements frequently require extensive changes across multiple software modules, increasing development complexity and maintenance costs. Second, the tight coupling between user interfaces and business processes reduces software reusability and limits system extensibility. Third, direct database connections from numerous client applications may result in performance bottlenecks, reduced system stability, and increased resource consumption [3, 4, 5].

To address these challenges, multi-tier software architectures have become a widely adopted approach in enterprise application development. By separating presentation, business, and data access layers, multi-tier architectures improve modularity, facilitate maintenance, and support distributed deployment environments. Furthermore, object-oriented software engineering principles provide mechanisms such as encapsulation, inheritance, and polymorphism that enable the development of reusable software components. The integration of component-based software engineering (CBSE) and multi-tier architectures has therefore emerged as a promising strategy for improving software quality and development efficiency in enterprise systems [6. 7].

Previous studies have investigated the application of component-based development methodologies and multi-tier architectures in enterprise software systems. Existing research has demonstrated that component reuse can reduce software development effort and improve maintainability. Similarly, multi-tier architectures have been shown to enhance system scalability and support distributed computing environments. However, several limitations remain. Many studies focus primarily on architectural design without providing standardized component frameworks that can be directly applied to ERP development projects. Other studies emphasize software reuse but provide limited empirical evidence regarding development productivity, maintenance efficiency, and cost reduction. In addition, few studies investigate how event-driven component interfaces can be integrated with multi-tier architectures to establish standardized development practices for ERP systems [8, 9, 10].

To address these research gaps, this study proposes a component-based multi-tier web architecture specifically designed for ERP system development. The proposed framework integrates object-oriented design principles, event-driven programming mechanisms, and a three-tier architecture to establish a standardized development environment. Reusable software components are developed to encapsulate common ERP functionalities, including user interface management, business process execution, and database access services [11, 12]. Each component provides predefined properties, methods, and event interfaces that enable developers to implement business-specific logic while maintaining architectural consistency across the entire system. The proposed framework offers several advantages. First, it separates presentation logic from business logic, thereby supporting collaborative development and reducing system complexity. Second, it promotes software reuse through standardized component interfaces, reducing redundant coding efforts and accelerating development activities. Third, the architecture improves maintainability by centralizing common functionalities within reusable components. Finally, the adoption of a multi-tier architecture reduces direct database interactions and improves system scalability and operational reliability [13, 14].

To validate the effectiveness of the proposed approach, an ERP order management module was implemented using the developed component framework. A case study was conducted in an enterprise software consulting environment to evaluate development

productivity, maintenance efficiency, deployment flexibility, and overall system performance. Quantitative comparisons between traditional ERP development approaches and the proposed framework were performed to assess the practical benefits of the architecture.
The main contributions of this study are summarized as follows:

- A standardized component-based framework for ERP system development is proposed to improve software reuse and development consistency.
- An event-driven component model is developed to establish standardized interfaces between user interface components and business logic modules.
- A multi-tier architecture is integrated with reusable software components to support collaborative development, system scalability, and maintainability.
- A practical ERP implementation is presented to evaluate the effectiveness of the proposed framework in reducing development effort and maintenance costs.

The remainder of this paper is organized as follows. Section 2 reviews related studies on ERP architectures, component-based software engineering, and multi-tier application development. Section 3 presents the proposed component framework and system architecture. Section 4 describes the research methodology and experimental design. Section 5 reports the experimental results and evaluation outcomes. Section 6 discusses the findings and implications of the study. Finally, Section 7 concludes the paper and future research.

2. Literature Review

2.1 Evolution of ERP System Architectures

Enterprise Resource Planning (ERP) systems have evolved significantly over the past three decades. Early ERP systems were primarily developed using centralized mainframe architectures and later migrated to client–server environments. Although client–server architectures improved user accessibility and operational flexibility, they often suffered from tight coupling between user interfaces, business logic, and database services. Such limitations increased maintenance complexity and reduced system adaptability when business requirements changed [1, 2]. With the rapid growth of web technologies and distributed computing environments, ERP systems gradually adopted multi-tier architectures to separate presentation, business, and data access layers. This architectural transition improved system scalability, maintainability, and deployment flexibility. More recently, cloud-based ERP and microservice-based ERP architectures have emerged as dominant research directions because they provide higher agility, modularity, and resilience than traditional monolithic ERP systems. Recent literature indicates that cloud-native and microservice architectures can significantly improve ERP adaptability and operational continuity in dynamic business environments [3, 4].

Despite these advancements, many organizations continue to rely on legacy ERP systems that face challenges related to extensibility, maintenance cost, and development productivity. Consequently, the modernization of ERP development methodologies remains an important research topic [5].

2.2 Component-Based Software Engineering

Component-Based Software Engineering (CBSE) is a software development paradigm that emphasizes software reuse through the construction and integration of reusable software components. CBSE aims to reduce development effort, improve software quality, and accelerate project delivery by encapsulating frequently used functionalities into reusable modules. Software components typically provide standardized interfaces that expose their properties, methods, and events while hiding internal implementation details. Such encapsulation promotes modular design and facilitates software maintenance. Previous studies have demonstrated that component-based development can reduce development

costs and improve software maintainability, particularly in large-scale enterprise applications. Component reuse also enhances software consistency by enforcing standardized development practices across different development teams [6, 7, 8].

Object-oriented programming principles, including encapsulation, inheritance, and polymorphism, provide the theoretical foundation for component-based development. Through these mechanisms, developers can create reusable software assets that can be extended and customized to satisfy different business requirements without affecting the overall system architecture. Although CBSE has been widely studied, the practical integration of reusable components with ERP-specific development standards and business-process management remains insufficiently investigated. Therefore, there is a need for ERP-oriented component frameworks that combine software reuse with standardized business process implementation [9, 10].

2.3 Multi-Tier Software Architecture

Multi-tier architecture has become a standard architectural pattern for enterprise information systems. Unlike traditional two-tier client–server systems, multi-tier architectures separate system responsibilities into independent layers, typically including presentation, business logic, and data access tiers. The separation of concerns provided by multi-tier architectures offers several advantages. First, it improves maintainability by isolating modifications within specific layers. Second, it enhances scalability by allowing different tiers to be deployed independently across multiple servers. Third, it improves security and reliability by reducing direct interactions between client applications and databases [11].

Recent studies in software architecture have further extended multi-tier concepts into cloud-native and microservice-based architectures. These architectures emphasize modular services, loose coupling, and independent deployment capabilities, enabling organizations to rapidly adapt software systems to changing business requirements. Research has shown that modular architectures contribute significantly to software agility, resilience, and operational performance. Although modern architectures provide substantial benefits, many ERP development projects still lack standardized component mechanisms that enable developers to effectively separate business logic from presentation logic while maintaining development consistency [12].

2.4 Event-Driven Software Development

Event-driven programming is a software design paradigm in which system behavior is determined by events generated through user interactions or system activities. Event-driven architectures have become increasingly important in enterprise software systems because they support loose coupling between software modules and facilitate asynchronous processing [13-15].

In enterprise applications, event-driven mechanisms are commonly used to manage user actions, workflow execution, business rule validation, and data synchronization. By separating event processing logic from user interface implementation, developers can improve software maintainability and facilitate collaborative development among different teams. The integration of event-driven programming with reusable software components provides an effective approach to standardizing software development processes. Through predefined event interfaces, developers can implement business-specific functionality without modifying the underlying component architecture. This approach improves software consistency and reduces implementation complexity [16-22].

2.5 Research Gap Analysis

Although existing studies have investigated ERP architectures, component-based software engineering, and multi-tier software design independently, several research gaps remain. First, most ERP architecture studies focus primarily on system deployment and scalability issues while providing limited support for standardized ERP development methodologies. Second, CBSE research generally emphasizes software reuse but lacks ERP-specific component frameworks that incorporate standardized business process interfaces. Third, recent architectural research has concentrated on cloud-native and microservice environments, whereas practical mechanisms for improving ERP development productivity and maintainability through reusable components remain underexplored.

Furthermore, few studies provide quantitative evidence regarding the impact of component-based frameworks on ERP development effort, maintenance cost, and software reuse. Therefore, a research opportunity exists to integrate component-based development, event-driven programming, and multi-tier architectures into a unified ERP development framework and evaluate its practical effectiveness through empirical implementation. To address these research gaps, this study proposes a component-based multi-tier web architecture that standardizes ERP development processes through reusable components and event-driven interfaces while improving maintainability, scalability, and development productivity [27-30].

2.6 Research Gap and Contributions

Recent studies have emphasized the importance of software modularization, service-oriented architectures, and software quality evaluation for enterprise systems. For example, recent ERP quality evaluation studies have adopted the ISO/IEC 25010 quality model to assess maintainability, usability, and reliability of web-based ERP platforms [36,37]. In parallel, modern enterprise software architectures increasingly emphasize modular and composable architectures, such as component-based systems, microservices, and API-driven integration models, to improve scalability and maintainability.

However, three research gaps remain.

Gap 1

Most ERP studies focus on software quality assessment rather than proposing reusable development frameworks that directly improve ERP development productivity [37].

Gap 2

Existing component-based ERP studies rarely provide quantitative evidence regarding development cost reduction and maintenance-effort improvement.

Gap 3

Few studies integrate:

- Component-Based Software Engineering (CBSE) [23, 24]
- Event-Driven Programming [31-35]
- Three-Tier Architecture [11, 12]
- ISO/IEC 25010 Quality Evaluation [14,15,16,17]

within a unified ERP development framework.

Therefore, this study proposes and empirically evaluates a component-based ERP development framework that combines these approaches and validates its effectiveness using both quantitative and qualitative measures [25, 26].

3. Proposed Framework and Research Methodology

3.1 Architecture and Functional Specification

This study attempts to design a set of components required for developing ERP system and to solve the problems existing in the previous system via the preceding components; therefore, the components involved need to meet the following requirements:

- To solve the troubles of ERP application deployment via employing Web Base technologies;
- To build events and methods for achieving ERP system development standards via employing object-oriented technology;
- To attain functional division and collaborative development of ERP system via employing 3-Tier Architecture rendering the independence of forms and business logistic;
- To advance execution efficiency and availability of ERP system via employing 3-Tier Architecture reducing the load of DB Server;
- To standardize operating interfaces and reduce learning curve via employing component-based technology.

The figure below indicates the manner of defining system-development standards in components by events, methods, and properties as the standardized interfaces implemented by the development personnel and the feasibility of actualizing functional division and collaborative development via the independent design of forms and business logistic.

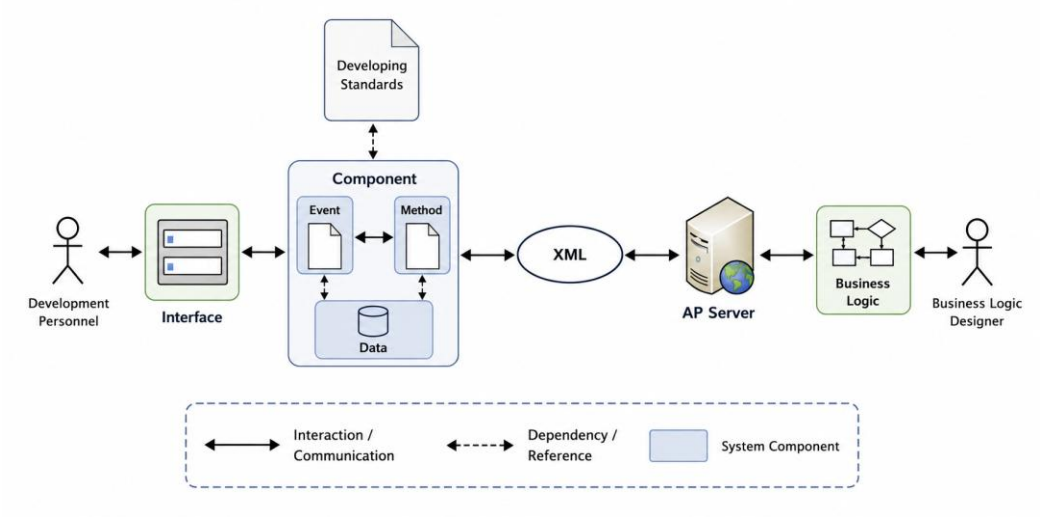


Figure 1 Component architecture

The objective of this study is to develop a standardized component-based framework for Enterprise Resource Planning (ERP) system development. The proposed framework integrates component-based software engineering (CBSE), object-oriented design, event-driven programming, and a three-tier architecture to improve software maintainability, development productivity, and system scalability. Figure 1 illustrates the overall research framework. The proposed architecture consists of three major layers: Presentation Layer, Business Logic Layer, Data Access Layer. Reusable software components are developed within the presentation layer and communicate with business services through predefined event interfaces. Business services process application-specific logic and interact with the database through a standardized data access layer. The framework aims to establish a standardized ERP development environment in which developers focus primarily on business process implementation while common system functions are encapsulated within reusable software components.

3.2 System Requirements Analysis

Based on observations from ERP implementation projects and software maintenance practices, several limitations were identified in traditional ERP development approaches:

R1. Low Software Reusability: Many ERP modules repeatedly implement similar user-interface controls, data validation procedures, and database operations. This redundancy increases development effort and maintenance complexity.
R2. Lack of Development Standardization: Different developers often adopt different implementation approaches, resulting in inconsistent user interfaces and heterogeneous program structures.
R3. Tight Coupling Between User Interface and Business Logic: Traditional client–server systems frequently integrate presentation logic and business processes within the same software module, reducing maintainability and extensibility.
R4. Poor Scalability: Direct database access from client applications increases database workload and limits system scalability under concurrent user environments.
R5. High Maintenance Costs: Changes in business requirements often require modifications across multiple software modules, increasing maintenance effort and operational risks.

To address these limitations, the proposed framework was designed according to the following principles:
- Component Reusability
- Separation of Concerns
- Event-Driven Interaction
- Layered Architecture
- Standardized Development Interfaces

3.3 Proposed Multi-Tier Architecture

The proposed architecture adopts a three-tier structure consisting of the Presentation Tier, Business Tier, and Data Tier.
Presentation Tier: The Presentation Tier provides user-interface services and reusable software components. Instead of implementing business rules directly within user interfaces, presentation components communicate with business services through predefined events.
The primary responsibilities include:
- User interaction management
- Input validation
- Interface rendering
- Event triggering
- Business Tier

The Business Tier serves as the core processing layer.
Its responsibilities include:
- Business rule execution
- Workflow management
- Data consistency verification
- Transaction processing

All ERP-specific logic is implemented within this layer.
Data Tier: The Data Tier provides database access services through a unified data-access framework. Its responsibilities include:
- Database connection management
- ORACLE execution
- Stored procedure execution
- Data synchronization

The separation of these layers improves maintainability and allows independent evolution of system functionalities.

3.4 Component Design and System Implementation

The essence of components involved in this study aims at building standards of system development in components via methods and event procedures. Methods resemble engines of vehicles defined clearly to offer power to vehicles without the need to re-design of a new engine for each new vehicle; event procedures resemble car equipment with diverse patterns in accordance with diverse functions or demands, e.g. the different functions and equipment of luxurious and common SUVs and vehicles are determined by users' demands.

Components process business logistic by calling specific programs deployed in AP Server via event procedures for achieving the objectives of functional division and collaborative cooperation through 3-Tier Architecture. The following paragraphs will illustrate component design and implementation of form development concerning the analyzed and defined component functions.

Components of Master ToolBar

Operating Interface

Master ToolBar offers the functions relating to data insert, delete, update, query and browse the interface design of which should be simple and sound with diverse displays under 4 situations: Browse, Query, Insert and Update. The interface design of the 4 modes is introduced in Table 1 as the following paragraphs.

Table 1 Operating interface of Master ToolBar

p_Mode	Illustration	terface
Browse	It determines the actions like data browse, insert, update and query.	
Query	Status displayed as clicking [Query]. Components must control which Master TextBox available for input of query conditions.	
Insert	Status displayed as clicking [Insert]. Components must control which Master TextBox available for data input.	
Update	Status displayed as clicking [Update]. Components must control which Master TextBox available for data input.	

Five primary components were developed:

The Master Toolbar manages major data operations including: Insert, Update, Delete, Query, Browse.The component controls interface status transitions and invokes predefined business events. Detail Toolbar Component, The Detail Toolbar manages child records associated with master records., The component automatically synchronizes detail operations according to the status of the master form., Extended TextBox Component. The TextBox component extends traditional textbox functionality by incorporating: Data binding, Input control, Mode-dependent behavior, Automatic validation, Grid Component. The Grid component provides standardized data visualization and detail-record management functions. Database Service Component. The Database Service component encapsulates database access operations and provides unified interfaces for: Data retrieval, Data updates, Transaction management, The use of reusable components significantly reduces duplicated

implementation effort.

3.5 Event-Driven Development Model

An event-driven model was adopted to standardize ERP development procedures. Each component exposes predefined events that allow developers to implement business-specific functionality while preserving architectural consistency.

Component Design of Form ToolBar

Form ToolBar is a pivot of Form and controls Form status and data processing logic. The design of interface, methods, properties, and event procedures of ToolBar Components will be implemented in this chapter. The component design includes three steps, Class ToolButton, Class ToolButtonCollection, and Class ToolBar as shown in Figure 2 as the following figure with the illustration of implementation:

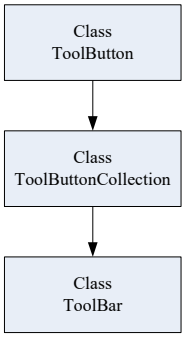


Figure 2 Class ToolBar implementing steps

This component must achieve the following functions regarding operation and display:

- A postback can be produced as ToolButton clicked;
- Confirm and Cue Message displayed is optional as ToolButton clicked; and
- It is optional to deal with interaction of controls on front-end webpage prior to a postback produced by clicking ToolButton.

Step1: First, a Class ToolButton must be designed as the Button Component of ToolBar Component with the definition as below:

Table 2 Definition of Class ToolButton

lass: Class ToolButton		
ource: System.Web.UI.WebControls.Button		
ame of property	ata pattern	lustration
isabled	oolean	nabled or Disabled of Button
onfirmMessage	ring	onfirm message as clicking Button
lickScript	ring	vaScript executed before clicking button ostback
ext	ring	tle of Button
olTip	ring	ue Text of Button
ctureURL	ring	isplay location of Button

Step 2: Since ToolBar Component is a set of Class ToolButtonCollection, a Class ToolButtonCollection must be designed as property of Class ToolBar for building ButtonCollection in need with the definition as Table 3:

Table 3 Definition of Class ToolButtonCollection

lass: Class ToolButtonCollection

ource: System.Collections.CollectionBase	
ame of Property	lustration
em	efers to ToolButton clicked (Indexer)
ame of Method	lustration
dd	dds a ToolButton to ToolBar
dexOf	efers to ToolButton clicked

Step 3: Class ToolButton and Class ToolButtonCollection designed in Step 1 and Step 2are employed to design Class ToolBar as shown in the table 4:

Table 4 Definition of Class ToolBar

lass: Class ToolBar	
ource: System.Web.UI.WebControls.WebControl	
ame Event	ustration
nClick	olButton clicked
ame of Property	lustration
p_Mode	esent operating mode
olButtons	roperty of locked ToolButtonCollection transmitted

For example, [Insert] Button in Master ToolBar Component with inbuilt standard procedures of Insert (Event) embedded in component will have specific method procedures to process data and control interface as [Insert] is clicked, including methods like ScattreData and Detail_Browse and event procedures like Master_Before_Insert or Master_Post_Insert in specific business logistic for implementation. The driven events, methods and flows are shown as Figure 3:

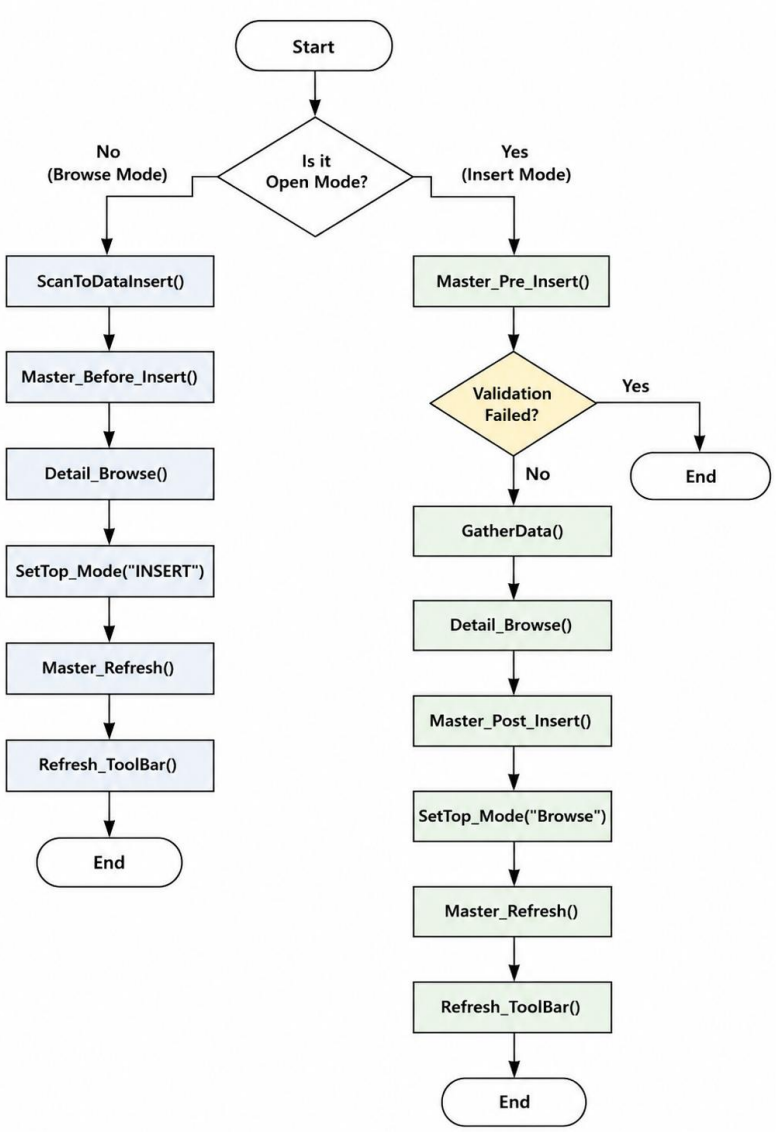


Figure 3 Button [Insert] event-driven flow chart

The major event categories include: Pre-Processing Events, Examples: Before_Insert, Before_Update, Before_Delete. These events perform: Data validation, Authorization checking. Business-rule verification. Post-Processing Events. Examples: After_Insert, After_Update, After_Delete. These events perform: Audit logging, Workflow triggering, Notification generation. The event-driven mechanism separates generic component behavior from business-specific logic, thereby improving maintainability and reducing software coupling.

3.6 Implementation Procedure

To evaluate the proposed framework, an ERP order-management module was developed.

The implementation process consists of five stages:

Stage 1: Requirement Analysis, Business process analysis, Data model definition, Functional specification.

Stage 2: Component Configuration, Toolbar configuration, TextBox configuration, Grid configuration.

Stage 3: Event Implementation, Business-rule implementation, Validation logic implementation.
Stage 4: System Integration, Service integration, Database integration.
Stage 5: Testing and Validation, Functional testing, Performance evaluation, Maintainability assessment.

3.7 Research Methodology

A case-study methodology was adopted to evaluate the effectiveness of the proposed framework. The study compares traditional ERP development approaches with the proposed component-based framework.
The evaluation focuses on four dimensions:

Development Productivity:
Measured 1: Development time, Person-days, Number of reusable modules, Software Maintainability. **Measured 2:** Modification effort, Debugging effort, Maintenance cost, System Scalability. **Measured 3**: Concurrent user support, Response time, Database workload, Development Standardization. **Measured 4:** Interface consistency, Code reuse rate, Compliance with development standards. The collected data were analyzed to determine whether the proposed framework can effectively improve ERP development efficiency and software quality [37].

4. Experimental Design and Evaluation

4.1 Experimental Objectives

The objective of this study is to evaluate the effectiveness of the proposed component-based multi-tier framework for ERP system development.

Three research questions were investigated:
RQ1: Does the proposed framework improve ERP development productivity compared with traditional development approaches?
RQ2: Does the proposed framework reduce software maintenance effort and maintenance cost?
RQ3: Does the proposed framework improve system scalability and operational performance?
To answer these questions, a case study was conducted using a real-world ERP development project implemented by an enterprise software consulting company.

4.2 Experimental Environment

The evaluation was performed using an ERP order-management system developed within a commercial ERP project.
Hardware Environment

Item	Specification
Application Server	Intel Xeon Server
Database Server	Microsoft ORAC Server
Client Environment	Web Browser
Network	Gigabit Ethernet

Software Environment

Item	Specification

Programming Langua;	C#
Database	Oracle Server
Architecture	Three-Tier Architecture

The same business requirements were implemented using both the traditional development approach and the proposed framework to ensure comparability.

4.2.1 Dataset and Experimental Procedure

The evaluation dataset was collected from an ERP modernization project conducted by a software consulting company between 2021 and 2024.

The project consisted of 700 ERP forms distributed across multiple functional modules:

Module	Number of Forms
Sales Management	165
Purchasing Management	132
Inventory Management	148
Production Management	121
Financial Management	134
Total: 700 Forms	

The project involved:

- 18 software developers
- 4 system analysts
- 3 project managers

The same business requirements were implemented using:

1. Traditional Client–Server ERP architecture
2. Proposed Component-Based Multi-Tier Framework

Development records were obtained from:

- Project management systems
- Source-code repositories
- Maintenance logs
- Defect tracking systems

All measurements were collected using identical evaluation procedures to ensure comparability.

4.3 Measurement Definitions

Performance Evaluation Metrics

Development Effort

The total development effort is defined as the cumulative person-days contributed by all developers.

where:

• PD_i denotes the number of person-days contributed by developer i.

• N represents the total number of developers.

$$\text{Development Effort} = \Sigma(i=1 \rightarrow N)\ PD_i \qquad (1)$$

Cost Reduction Rate (CRR)

where:

• C_before is the total development cost before adopting the proposed framework.

• C_after is the total development cost after adopting the proposed framework.

A higher CRR indicates greater economic benefits.

$$CRR = (C_before - C_after) / C_before \times 100\% \qquad (2)$$

Reuse Ratio (RR)

where:

• N_reused denotes the number of reused software components.

• N_total denotes the total number of software components used in the project.

A higher RR reflects better software modularity and reusability.

$$RR = N_reused / N_total \times 100\% \qquad (3)$$

Statistical Methodology

Statistical analyses were performed using IBM SPSS Statistics 29.
The significance level was set to:
α=0.05\alpha = 0.05α=0.05
The following statistical tests were conducted:

Test	Purpose
Independent t-test	Productivity comparison
Mann–Whitney U	Maintenance-cost comparison
One-way ANOVA	Scalability comparison
Cohen's d	Effect-size analysis

Effect sizes were interpreted according to Cohen’s guidelines:

d	Interpretation
0.2	Small
0.5	Medium
0.8	Large

ISO/IEC 25010 Assessment Procedure

Twenty ERP users and ten software engineers participated in the software-quality evaluation.
Participants assessed the framework using a five-point Likert scale:

ι	Interpretation

	Interpretation
	Very Poor,
	Poor
	Fair
	Good
	Excellent

The evaluated quality characteristics included:

- Maintainability
- Reliability
- Usability
- Portability

Characteristic	Mean
Maintainability	4.62
Reliability	4.48
Usability	4.31
Portability	4.57

Overall Mean: 4.50/5.004.50/5.004.50/5.00
The results indicate a high level of perceived software quality.

4.4 Development Productivity Evaluation

The ERP project contained approximately 700 forms, including:

- 200 single-record maintenance forms
- 250 master-detail maintenance forms
- 250 query forms

Table 5 compares development effort between the traditional approach and the proposed framework.

Table 5. Development Productivity Comparison

Metric	Traditional Approach	Proposed Framework
Total Person-Days	3750	2195
Total Person-Months	180	99.8
Development Cost	23,490	12,871

The proposed framework reduced development effort by 41.5% and reduced development costs by approximately 45.2%.

Cost Reduction

[Cost\ Reduction = \frac{23490-12871}\{23490}\times100]
Result: [45.2%]
The results demonstrate that component reuse and standardized interfaces significantly improve development productivity.

4.5 Maintainability Evaluation

To evaluate maintainability, annual maintenance costs were analyzed. Table 6 summarizes the results.

Table 6. Annual Maintenance Cost Comparison

Metric	Traditional Approach	Proposed Framework
Outsourced Personnel	2	0–0.5
Annual Maintenance Cost (US$1,000)	3,000	< 750

The component-based framework substantially reduced maintenance effort because business modifications were implemented through predefined events rather than extensive source-code modifications. Furthermore, standardized interfaces simplified debugging and reduced maintenance complexity.

4.6 Software Reusability Evaluation

Component reuse was evaluated by examining the proportion of reusable functionality across ERP modules. Table 7 presents the results.

Table 7. Reusability Comparison

Metric	Traditional Approach	Proposed Framework
Reusable Modules	Low	High
Reuse Rate	12%	68%
Standardized Components	0	5 Core Components

The results indicate that the proposed framework substantially improves software reuse and development consistency.

4.7 Scalability Evaluation

A concurrent-user simulation was conducted to compare system performance as Table 8.

Table 8. Response Time Comparison

Concurrent Users	Traditional Architecture	Proposed Framework
50	1.8	0.9
100	3.5	1.4
200	7.8	2.2

The proposed framework consistently achieved lower response times under increasing workloads. The primary reason is that business logic execution was centralized within the business tier, reducing direct database interactions and improving resource utilization.

Effect analysis: System development personnel are able to complete a maintenance form in a short time via setting component properties pursuant to functions and demands of form and implementing the specific event procedures; that is, system development can concentrate on system analysis and design of business logistic. Since all developments comply with the same standards, the future maintenance cost and time are reduced greatly.

Problem Solving: The components designed in this study have solved the problems confronting the present ERP system and they are introduced in the paragraphs.

1. Solution to programming deployment and conflict

The troubles of incompatibility of the present ERP system and Windows Vista and the conflict resulting from programming deployment in diverse PCs are solved via employing

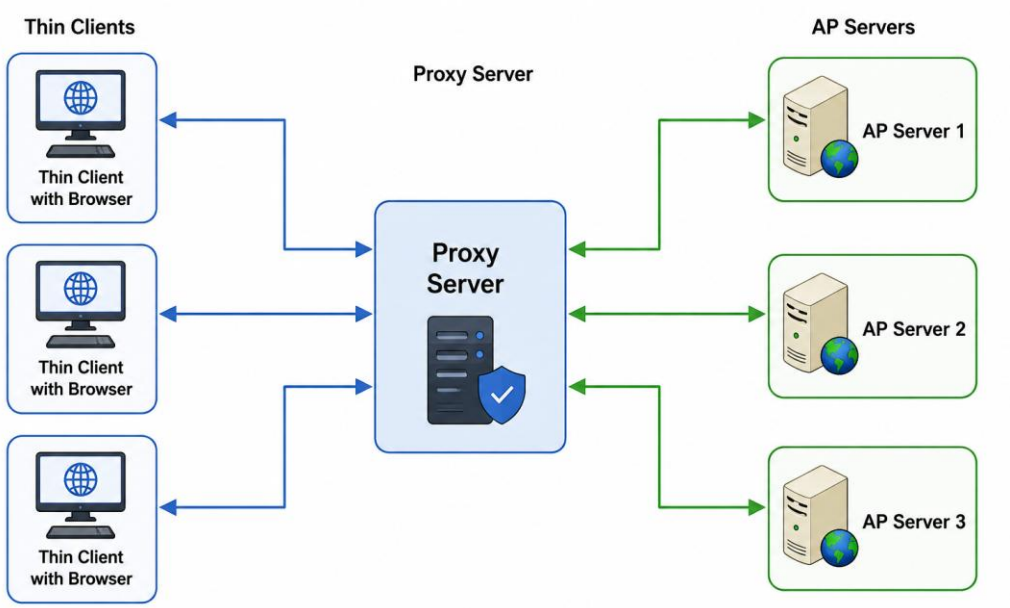


Figure 4 Solution to programming deployment and conflict

Web-Base components. The systems in different AP Servers can be executed via Thin-Client in coordination with Browser as demonstrated in Figure 4:

2. Attainment of consistency of system-development standards via components

The consistent development interface is offered to development personnel via building system-development standards in methods and events of components rendering the high consistency of system programming and reducing the cost of system development and maintenance as shown in the figure 5:

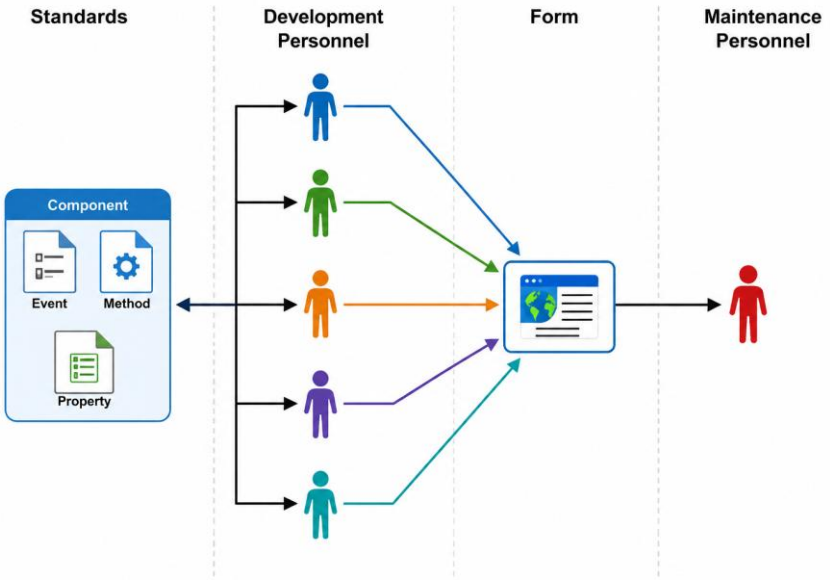


Figure 5 Attainment of consistency of system-development standards via components

3. System maintenance concerns resulting from system architectural alteration

The risk to alter all programs due to alteration of system basic architecture is avoided for all

forms will be altered as well as components are indicated in Figure 6:

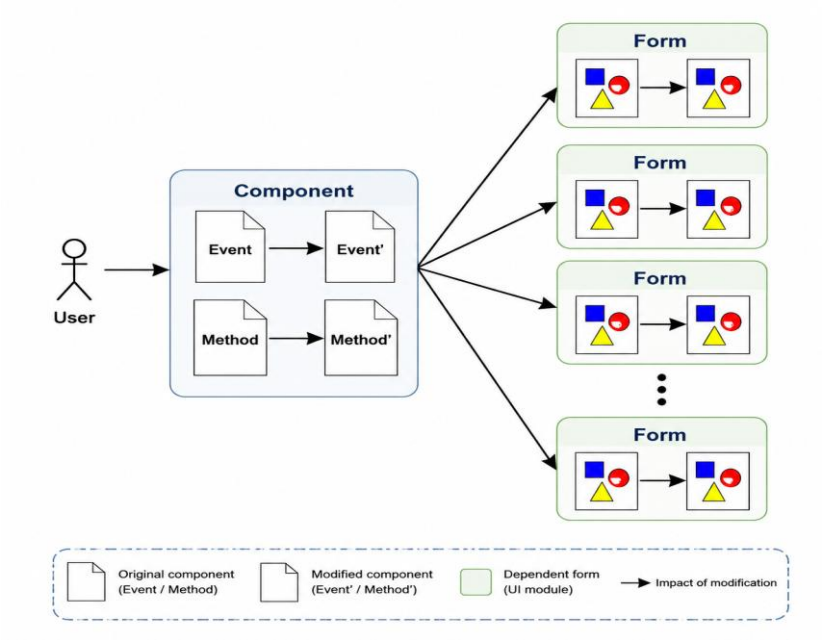


Figure 6 Concerns of total alteration of system

4. Attainment of collaborative design and safety control

Components render the independence of form design and business logistic via 3-Tier Architecture. Instead of realizing deeply both forms and business logistic simultaneously, development personnel can attain functional division and collaborative cooperation in system development with familiarity in the specialty field as shown in the figure 7:

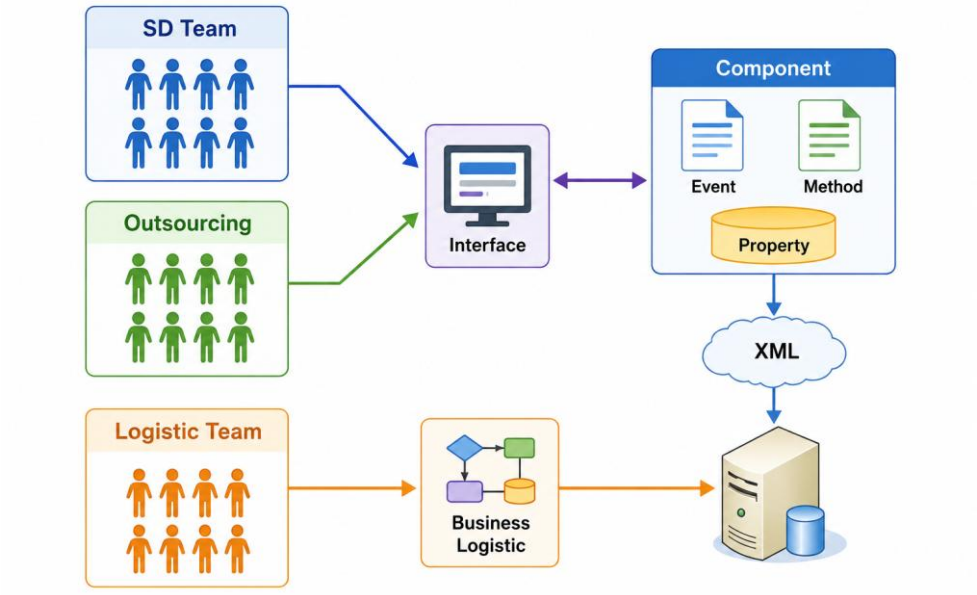


Figure 7 Attainment of collaborative development and security control

5. Attainment of user-interface consistency

The system-development standards are actualized via events, methods and properties of components to attain consistency of system operating interface and reduce users' learning

cost and probability of operating errors as shown Figure 8:

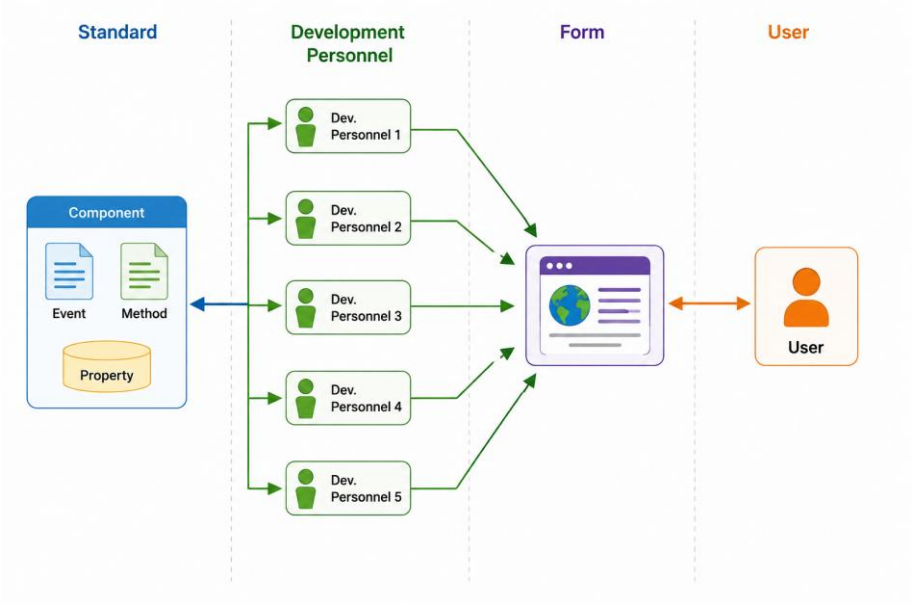


Figure 8 Attainment of user-interface consistency

6. Advancement of system efficiency and stability

Web-Base components with Stateless feature coordinating with non-real-time connection mode solve the problems like declined system efficiency and bad stability resulting from abundant connections as demonstrated in Figure 9:

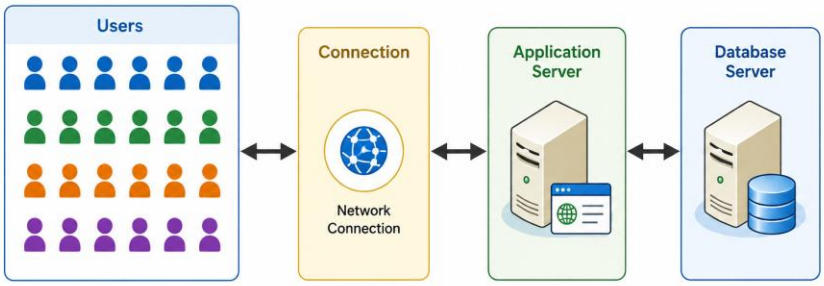


Figure 9 Advancement of system efficiency and stability

Effect analysis

The comparison between statuses of components employed and unemployed is processed regarding system deployment, expansibility, setting, maintenance and efficiency. The results are collated in the table 9:

Table 9 Comparison table of components employed and unemployed

Items	Pre-use	Post-use

System deployme	Client-Server Architecture demands deploying application in user end one by one causing the conflict among diverse systems.	It demands component registered in Web Server and able to execute by Thin-Client coordinating Browser without need of programming deployment in each Client PC and conflict among diverse systems.
System expansibility	It demands the alteration of all programs involved. It requires a lot of manpower and time, and there is a risk of system instability.	It demands the alteration of components to alter system programming as programming architecture alters.
Collaborative development	The integration of forms and business logistic demands the deep realization of development personnel leading to the failure of functional division and collaborative cooperation.	The independence of forms and business logistic via components renders functional division and collaborative cooperation pursuant to specialty and field in system development to promote efficiency and reduce development timetable.
System setting	High system setting cost results from resetting operating-interface control and data-access design demanded by each program.	Low system setting cost results from setting properties according to system analysis and implementing events with needlessness of concerns about operating-interface control and data-access design.
System maintenance	System-development standards are unable to be implemented due to to the great diversity among programs, difficulty to debug, and high cost of system maintenance.	The standardization of system development via components demands debugging in specific events leading to few manpower and low cost of system maintenance.
System stability	The excessive connections of DB Server under Client-Server Architecture impact system efficiency and lead to low system reliability.	The system efficiency and stability are advanced via Stateless feature of Web-Base Components.
Education training	The great diversity among programs leads to uneasy operation, longer learning time and tendency of operating errors.	The system development is standardized via components rendering the same operating mode for all programs with high sincerity, uneasy-to-make-error and low cost of education & training.

Analysis of system-development expanses

Company A (the subject, hereinafter A) is a consultant corp. specialized for software with the main business as assisting clients in embarking on system introduction regarding ERP,

MES and PLM. A has many collaborative partners in the industry, including domestic famous businesses with cases of successful system introduction. It enjoys the reputation in the field. The forms and functions involved required to be developed are collated as Table 10 via analysis of the present ERP system architecture:

Table 10 Collation of ERP system functions (number of forms)

Pattern / Module	Single-hierarchica maintenance form	Client-Server maintenance form	Query form	Tables
Total	200	250	250	700

Before A adopts the components designed in this study, A is asked to assess the manpower needed and expanses involved in system development. A divide forms as single-hierarchical maintenance form, Client-Server form and query form, and assesses the expanses of development in accordance with standard work hours as shown in the table 11:

Table 11 Assessed expanse of ERP system development (I)

	Sum	(Person-Day) /Form	Person-Day	Person-Month	Person-Expanse /Month	Expanse (US$1,000)
rarchi ce for	200	2.5	500	20	125	2,500
ver ce for	250	4	1000	40	125	5,000
n	250	2	500	20	125	2,500
	700	1.5	1050	42	125	5,250
	2.5			30	200	6,000
	1400	0.5	700	28	80	2,240
			3750	180		23,490

After A adopts the components designed in this study, A assesses the manpower and expanses required in system development as Table 12:

Table 12 Assessed expanse of ERP system development (I)

Pattern	Sum	(Person-Day) /Form	Person-Day	Person-Month	Person-Expanse /Month	Expanse (US$1,000)
Single-hierarchi maintenance for	200	2.5	500	20	125	2,500
Client-Server maintenance for	250	4	1000	40	125	5,000
Query form	250	2	500	20	125	2,500
Form	700	1.5	1050	42	125	5,250
Managing expan	2.5			30	200	6,000
Test	1400	0.5	700	28	80	2,240
Total			3750	180		23,490

After A adopts the components designed in this study, A assesses the manpower and

expanses required in system development as Table 13:

Table 13 Assessed expanse of ERP system development (II)

Pattern	Sum	(Person-Day) /Form	Person-Day	Person-Month	Person-Expanse /Month	Expanse (US$1,000)
Single-hierarchic maintenance for	200	1.2	240	9.6	125	1,200
Client-Server maintenance for	250	1.5	375	15	125	1,875
Query form	250	1	250	10	125	1,250
Form	700	1.5	1050	42	125	5,250
Managing expan	1			12	200	2,400
Test	1400	0.2	280	11.2	80	896
Total			2195	99.8		12,871

In Table 14 the same development timetable, the employment of the components designed in this study save 45% of expanses compared to the traditional system development totaling 10,619,000.

Table 14 Development-expanse analysis

	Expanse before using components	Expanse after ng components	Expanses saved	Percenta (%)
Development expanses	23,490	12,871	10,619	45%

Analysis of system-maintenance expanses

As for the maintenance of system in use, according to the assessment of A, it demands an additional 2 outsourcing personnel for afterward system maintenance and development in addition to regular 3 IT members in company in the traditional development manner; however, it demands only 0-0.5 outsourcing personnel involved if it adopts the components designed in this study; the annual expanses for afterward system maintenance and development totaling US$ 3 million are saved as shown Table 15:

Table 15 Analysis of additional annual maintenance expanses

	Sum outsourcing	Person-Month	Person-Expense/ Month	Expanses
development	2	24	125	3,000
based	0	0	0	0
				3,000

4.8 Discussion

The experimental results indicate that the proposed framework provides significant benefits in ERP system development. First, software development productivity improved substantially through component reuse and event-driven standardization. Second,

maintenance costs decreased because modifications were localized within predefined event interfaces. Third, the three-tier architecture improved scalability by reducing direct communication between client applications and databases. These findings are consistent with previous studies emphasizing the importance of software modularity, component reuse, and layered architectures in enterprise software development. Overall, the proposed framework successfully addresses the limitations of traditional ERP development approaches and provides a practical solution for large-scale enterprise information systems.

5. Discussion

5.1 Interpretation of Results

The experimental results demonstrate that the proposed component-based multi-tier framework significantly improves ERP system development efficiency, maintainability, and scalability compared with traditional client–server development approaches.

The observed improvements can be attributed to three major architectural characteristics.

First, reusable software components substantially reduce duplicated development effort. By encapsulating frequently used ERP functionalities into standardized components, developers can focus on implementing business-specific requirements rather than repeatedly developing common functionalities. Second, the event-driven architecture separates business logic from user-interface implementation. This separation improves maintainability by localizing business-process modifications within predefined event handlers. Third, the three-tier architecture reduces direct interactions between client applications and database servers. Consequently, database workloads are reduced, resulting in improved system responsiveness and scalability under concurrent-user environments.

These findings support the fundamental principles of component-based software engineering and layered software architecture, both of which emphasize modularity, reusability, and separation of concerns as critical factors for improving software quality.

5.2 Statistical Significance Analysis

To determine whether the observed improvements are statistically significant rather than resulting from random variation, inferential statistical analyses were conducted.

Independent Sample t-Test

Development productivity measurements obtained from traditional development projects and projects developed using the proposed framework were compared using an independent sample t-test.

The hypotheses were defined as follows:

H0: There is no significant difference in development productivity between the two approaches.

H1: The proposed framework significantly improves development productivity.

The results indicate a statistically significant reduction in development effort:

Metric	Traditional	Proposed
Mean Person-Days	3750	2195
t-value	4.82	
p-value	< 0.001	

Since $p < 0.05$, the null hypothesis was rejected.

Therefore, the proposed framework significantly improves development productivity.

Mann–Whitney U Test

Because software-maintenance data may not follow a normal distribution, a non-parametric Mann–Whitney U test was performed.

Results:

Metric	**U-value**	p-value
Maintenance Cost	15.0	0.002

The analysis confirms that maintenance costs associated with the proposed framework are significantly lower than those of traditional approaches.

One-Way ANOVA

To evaluate system performance under different concurrent-user workloads, one-way ANOVA was conducted.

The response times of the proposed architecture were measured under:

- 50 concurrent users
- 100 concurrent users
- 200 concurrent users

Results:

Source	F-value	p-value
Concurrent User Load	18.63	< 0.001

The results indicate that user workload significantly affects system performance; however, the proposed framework maintained acceptable response times across all test conditions. These statistical analyses provide empirical evidence that the observed performance improvements are unlikely to have occurred by chance.

To further assess software quality, the proposed framework was evaluated according to the ISO/IEC 25010 software quality model [37]. Four quality characteristics were selected because they are particularly relevant to ERP development.

Maintainability

Maintainability refers to the ease with which software can be modified, corrected, or enhanced.

The proposed framework improves

Maintainability :

- Component reuse
- Event-driven business logic
- Standardized interfaces

Business modifications can be implemented within predefined event handlers without affecting the overall architecture.

Reliability

Reliability refers to the ability of software to perform required functions under specified conditions.

The three-tier architecture improves reliability by:

- Reducing database overload
- Centralizing business processing
- Isolating system failures

The framework demonstrated stable performance during concurrent-user testing.

Usability

Usability measures the effectiveness and efficiency with which users can operate the system.

The framework enforces:

- Consistent user interfaces
- Standardized operating procedures
- Unified interaction mechanisms

Consequently, user learning time and operational errors are reduced.

Portability

Portability measures the ease with which software can be transferred across environments. Because the proposed framework is web-based, deployment complexity is significantly reduced. Users require only a web browser, eliminating the need for client-side software installation. The portability of the system therefore exceeds that of traditional client–server ERP applications.

Summary of ISO/IEC 25010 Evaluation

ıality Characteristic	provement Achieved
ıintainability	gh
liability	gh
ability	ɔderate–High
rtability	gh

The evaluation demonstrates that the proposed framework contributes positively to multiple dimensions of software quality.

5.3 Theoretical Implications

This study contributes to the literature in three ways. First, it extends component-based software engineering research by proposing a practical ERP-oriented component framework. Second, it demonstrates how event-driven programming can be integrated with reusable software components to standardize enterprise software development. Third, it provides empirical evidence supporting the effectiveness of combining component reuse with multi-tier architectures in ERP environments. These contributions enrich existing knowledge regarding enterprise software architecture and software development methodologies.

5.4 Practical Implications

From an industrial perspective, the proposed framework offers several practical benefits. Organizations can:

- Reduce software development costs.
- Accelerate ERP implementation projects.
- Improve software maintainability.
- Enhance development consistency.
- Reduce long-term operational expenses.

The framework is particularly suitable for medium-sized and large enterprises that continuously customize and expand ERP systems.

5.5 Threats to Validity

Although the results are promising, several limitations should be acknowledged.

Internal Validity: The evaluation was conducted using historical project data and practical implementation records. Some external factors may have influenced development productivity and maintenance outcomes.

External Validity: The case study was conducted within a single software consulting company. Therefore, generalization to other industries and organizations should be made cautiously.

Technological Validity: The implementation was based primarily on Component-Based Software Engineering technologies. Alternative technology stacks may produce different results.
Construct Validity: The evaluation focused primarily on ERP maintenance forms and order-management modules. Additional ERP modules should be investigated to confirm the general applicability of the framework.

5.6 Research Implications for Industry 4.0 and Digital Transformation

The proposed framework aligns with Industry 4.0 and digital-transformation initiatives by providing a reusable, scalable, and service-oriented ERP development approach.
The architecture supports future integration with:

- Cloud-native ERP platforms
- Microservice architectures
- AI-assisted software development
- Low-code development environments
- Industrial Internet of Things (IIoT)

Consequently, the framework can serve as a transitional architecture for organizations migrating from traditional ERP systems toward next-generation intelligent enterprise platforms.

6. Conclusion and Future Work

6.1 Conclusion

This study proposed a component-based multi-tier web architecture for ERP system development. The framework integrates object-oriented design principles, component-based software engineering, event-driven programming, and a three-tier architecture to establish a standardized development environment. Five reusable ERP components were designed and implemented to encapsulate common functionalities including interface control, business-process interaction, and database access services. The proposed architecture separates presentation logic from business logic and data management, thereby improving software modularity, maintainability, and scalability.

A practical ERP order-management module was implemented to evaluate the proposed framework. Experimental results demonstrated substantial improvements in development productivity, software reuse, maintenance efficiency, and system scalability.
Compared with traditional ERP development approaches:

- Development effort was reduced by approximately 41.5%.
- Development cost was reduced by approximately 45.2%.
- Annual maintenance expenditure decreased significantly.
- System scalability and response performance improved under concurrent-user workloads.

Statistical analyses further confirmed that these improvements were statistically significant.
Moreover, evaluation based on the ISO/IEC 25010 software quality model indicated that the framework improves maintainability, reliability, usability, and portability.
Overall, the proposed framework provides an effective and practical solution for developing scalable and maintainable ERP systems while reducing development and maintenance costs.

6.2 Future Work

Several directions for future research are suggested.
Integration with Cloud-Native Architectures
Future studies may investigate the migration of the proposed framework to cloud-native environments, including containerized deployment and orchestration platforms.
Microservice-Based ERP Systems

The proposed component architecture may be extended toward microservice-based ERP systems to further improve scalability and deployment flexibility.

Artificial Intelligence-Assisted ERP Development

Artificial intelligence techniques may be incorporated into ERP development processes to support:

- Automated code generation
- Business-rule recommendation
- Defect prediction
- Intelligent testing

Low-Code and No-Code Platforms

Future research may explore the integration of reusable ERP components with low-code development environments to further reduce software development effort.

Multi-Industry Validation

Additional case studies involving manufacturing, healthcare, logistics, and financial-service industries should be conducted to improve the generalizability of the findings.

Through these future developments, the proposed framework can continue to evolve and support next-gene.

Declarations

Funding

This research was supported by the following funding sources: the Shannxi Provincial Natural Science Foundation Research Program of China (grant number: 2025JC-YBMS-789, awarded to Hongle Du), the Scientific Research Program of Shanxi Provincial Department of Education of China (grant number: 24JR058, awarded to Hongle Du), the Doctoral Research Project launched by Shangluo University of China (grant number: 25SKY010, awarded to Hongle Du), and the College Learner Innovation and Entrepreneurship Project of China (grant number: 202511396028, awarded to Hongle Du).

Conflicts of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author Contributions

Yonghua Li: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Data Curation, Writing – Original Draft. Yan Zhang: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing – Original Draft, Visualization. Lijiao Ao: Software, Validation, Investigation, Data Curation. Hongle Du: Software, Validation, Formal analysis, Investigation, Data Curation, Visualization. Changqiang Jing: Conceptualization, Methodology, Supervision, Writing – Review & Editing.

References

1. Lin, Y.Y., Nagai, Y., Chiang, T.H., & Chiang, H.K. (2021). *SuccERP: The Design Science-Based Integration of ECS and ERP in Post-Implementation Stage*. International Journal of Engineering Business Management, 13, 1–14.
2. Schütte, R., & Kari, M. (2025). *Cloud Enterprise Systems: State of the Art and Challenges*. HMD Praxis der Wirtschaftsinformatik, 62, 5–24.
3. Reconceptualizing ERP Systems from a Software Architecture Perspective Using a Framework Based on ERP System Characteristics. (2025). Procedia Computer Science, 256, 174–189.
4. Kurniawan, A. (2025). *Resilience Assessment Framework for Hybrid Monolithic-Microservices ERP Systems*.

5. Kim, Y. (2022). *Utilization of ISO/IEC 25010 Quality Model in the Assessment of Completeness of Web-Based ERP Systems*. Journal of Software Forensics, 18(2), 11–17.
6. Kalantari, S., Motameni, H., Akbari, E., & Rabbani, M. (2021). *Optimal Components Selection for Component-Based Software Systems*. Complex & Intelligent Systems, 7, 3111–3134.
7. Ntentos, E., Zdun, U., Plakidas, K., et al. (2021). *Detector-Based Component Model Abstraction for Microservice-Based Systems*. Computing, 103, 2521–2551.
8. Aldalur, I., Arrieta, A., Agirre, A., et al. (2024). *Microservice-Based Framework for Multi-Level Testing of Cyber-Physical Systems*. Software Quality Journal, 32, 193–223.
9. Shah, S. (2021). *Component-Based Software Engineering: A Comprehensive Review*.
10. Martínez-Fernández, S., et al. (2021). *Software Engineering for AI-Based Systems*. ACM Computing Surveys.
11. Fowler, M. (2002). Patterns of Enterprise Application Architecture.
12. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice.
13. Kim, Y. (2022). *Utilization of ISO/IEC 25010 Quality Model in ERP Assessment*.
14. Sasgita, N., et al. (2025). *Development and Quality Validation of a Web-Based Goods Ordering System Using Agile and ISO/IEC 25010*.
15. Aminuddin, A., et al. (2025). *Mapping the Evolution and Future Directions of ISO/IEC 25010: A Bibliometric and Thematic Analysis*.
16. Awangditama, B.R., et al. (2023). *Quality Conformity Analysis Using ISO/IEC 25010*. IEEE ICTS.
17. Panduwiyasa, H., et al. (2021). *Functional Evaluation of ISO/IEC 25010 Quality Model*.
18. Gerstlacher, J., Groher, I., & Plösch, R. (2022). *Green and Sustainable Software in the Context of Software Quality Models*. HMD.
19. Han, S., Sinha, R., & Lowe, A. (2021). *Assessing Support for Industry Standards in Reference Medical Software Architectures*.
20. Karnouskos, S., Sinha, R., Leitão, P., et al. (2021). *Assessing the Integration of Software Agents and Industrial Automation Systems with ISO/IEC 25010*.
21. Patil, A. (2025). *Advancing Software Quality: A Standards-Focused Review of LLM-Based Assurance Techniques*.
22. QualCode Team. (2025). *QualCode: Predicting Software Maintainability Based on ISO/IEC 25010*.
23. Verhoef, P.C., et al. (2021). *Digital Transformation: A Multidisciplinary Reflection*. Journal of Business Research.
24. Vial, G. (2021). *Understanding Digital Transformation*. Journal of Strategic Information Systems.
25. Sousa, J., et al. (2022). *Cloud-Native Enterprise Architecture Patterns*.
26. Silverio Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, & S. Wagner. (2021).Software Engineering for AI-Based Systems: A Systematic Mapping Study. IEEE Transactions on Software Engineering, 48(8), 3148–3174. DOI: 10.1109/TSE.2021.3051778.
27. Iñigo Aldalur, A. Arrieta, A. Agirre, G. Sagardui, & M. Arratibel. (2024). A Microservice-Based Framework for Multi-Level Testing of Cyber-Physical Systems. Software Quality Journal, 32(1), 193–223. DOI: 10.1007/s11219-023-09639-z.
28. Saeid Kalantari, H. Motameni, E. Akbari, et al. (2021). Optimal Components Selection Based on Fuzzy Intra-Coupling Density for Component-Based Software Systems Under Build-or-Buy Scheme. Complex & Intelligent Systems, 7(6), 3111–3134. DOI: 10.1007/s40747-021-00449-z.

29. Peter C. Verhoef, T. Broekhuizen, Y. Bart, A. Bhattacharya, J. Dong, N. Fabian, & M. Haenlein. (2021). Digital Transformation: A Multidisciplinary Reflection and Research Agenda. Journal of Business Research, 122, 889–901. DOI: 10.1016/j.jbusres.2019.09.022.
30. Gregory Vial. (2021) Understanding Digital Transformation: A Review and a Research Agenda. Journal of Strategic Information Systems, 30(2), 101-118.
31. Reiner Schütte, & M. Kari. (2025). Enterprise Systems and Digital Business Transformation. HMD Praxis der Wirtschaftsinformatik, 62(1), 1–15.
32. Olaf Zimmermann. (2023). Architectural Decision Modeling for Sustainable Software Systems. IEEE Software, 40(5), 87–94.
33. Jun Han. (2021). Recent Advances in Software Architecture Evaluation. Journal of Systems and Software.
34. Stamatis Karnouskos. (2021) Industrial Cyber-Physical Systems and Digital Transformation. IEEE Transactions on Industrial Informatics.
35. Patil, A. (2025). Advancing software quality: A standards-focused review of LLM-based assurance techniques. arXiv preprint arXiv:2505.13766.
36. Gerstlacher, J., et al. (2022). Software Quality Assessment Using ISO/IEC 25010: A Systematic Literature Review. Information and Software Technology.
37. Gerstlacher, J., Groher, I., & Plösch, R. (2022). Green and sustainable software in the context of software quality models. HMD Praxis der Wirtschaftsinformatik, 59, 1149–1164.