Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Modernizing Legacy Enterprise Configuration Interfaces: A Bootstrap-Based Approach to Oracle Configurator UI Enhancement

Nitin Thind
Punjabi University, India

ARTICLE INFO

Received: 12 March 2026
Revised: 05 April 2026
Accepted: 28 April 2026

ABSTRACT

User experience evaluation is critical for identifying user requirements and mitigating the risk of product failure during the development of interactive systems. This article offers a thorough framework for updating Oracle Configurator user interfaces using a Bootstrap-based method that keeps the backend infrastructure intact while providing modern frontend experiences. The persistence of legacy web infrastructures and database applications poses major obstacles to organizational agility, security, and maintainability. Outdated enterprise configuration systems lack modern security patches, responsive design, and integration capabilities, leading to operational inefficiencies and elevated risk. This article presents a four-step plan for updating systems that includes keeping the basic structure, redesigning the user interface, and adding interactive features based on the framework. The framework addresses three key challenges prevalent in enterprise software environments: diversified user goals and operational patterns, limited access to external users during development, and the necessity for long-term evaluation and maintenance. By carefully examining current Oracle Configurator designs and modern web development methods, this study shows how companies can greatly speed up configuration tasks, increase user involvement, and boost overall system performance. The dual-panel layout design featuring selection interfaces and real-time summary displays represents a significant advancement over traditional single-page configurator layouts. Post-implementation results from similar modernization initiatives demonstrate reductions in administrative overhead, improvements in security posture, and enhanced form load performance. The successful implementation validates the framework as a robust, scalable blueprint for organizations seeking to enhance their digital capabilities while preserving business logic and data integrity embedded in legacy Oracle systems.

Keywords: Oracle Configurator, User Interface Modernization, Bootstrap Framework, Legacy System Transformation, Enterprise Configuration Management

1. INTRODUCTION

1.1 Background and Problem Statement

In today's digital world, an organization's ability to adapt is directly related to how up-to-date its software infrastructure is. Legacy systems, defined as information systems that are technologically obsolete yet critical to daily operations, present a formidable paradox in that they are both vital and vulnerable [4]. Enterprise configuration systems play a pivotal role in sales enablement processes, yet many organizations continue to rely on outdated user interfaces that fail to engage sales personnel effectively and result in prolonged configuration times. The Oracle Configurator, while powerful in its backend capabilities, often presents users with interfaces that do not meet contemporary usability standards, creating friction in the sales process and reducing overall productivity. System resilience metrics show that availability drops by fifty-three percent during busy times, with recovery taking an average of 5.2 hours instead of the thirty-seven minutes seen in updated systems, and this difference negatively affects business results, leading to customer satisfaction scores that are twenty-three percent lower for companies stuck with old platforms.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1.2 Legacy System Constraints and Business Impact

Legacy systems constrain business agility by restricting deployment frequency to once per month, compared to the weekly or daily deployments possible with modernized architectures [1]. These constraints manifest across multiple dimensions of organizational performance, affecting not only technical operations but also business outcomes and competitive positioning. Maintenance costs increase at an annual rate of 18.5 percent for systems older than eight years, consuming approximately seventy-two percent of IT budgets rather than enabling innovation [1]. Monolithic applications typically experience 4.3 times longer development cycles, with average feature delivery timeframes extending to six to eight months versus four to six weeks for microservices-based alternatives [1]. Technical debt compounds at an alarming rate in monolithic systems, with code maintainability declining by approximately 17.8 percent annually after the first five years of development [1]. Legacy systems accumulate an average of 2.3 technical debt issues per hundred lines of code, with particularly high concentrations in authentication mechanisms, data access layers, and transaction management components that exhibit 3.7 times higher defect rates and require 4.2 times longer to modify than comparable clean code segments [1].

1.3 Research Objectives and Scope

This research investigates front-end modernization strategies that preserve backend infrastructure integrity while delivering contemporary user experiences. The primary objective is to develop and validate a comprehensive framework for modernizing Oracle Configurator user interfaces using Bootstrap and modern web technologies. When more than five thousand users try to use legacy applications at the same time, the response time can slow down by 215 to 340 percent, which means we need to upgrade the system, leading to extra costs of about twelve thousand to eighteen thousand dollars for each added capacity unit. Integration capabilities present equally concerning metrics, with sixty-eight percent of legacy applications lacking standardized API interfaces and requiring custom integration code that extends project timelines by an average of 156 percent compared to API-first approaches [1]. Organizations maintaining legacy identity and authorization systems experience a thirty-seven percent higher rate of security incidents, with investigation and remediation cycles averaging 18.5 days compared to 4.7 days for modernized architectures [1].

Metric	Value
Annual Code Maintainability Decline	17.8%
Technical Debt Issues per 100 Lines of Code	2.3
Defect Rate Increase in Debt-Heavy Sections	3.7×
Modification Time Increase in Debt-Heavy Sections	4.2×
Legacy Applications Lacking Standardized APIs	68%
Project Timeline Extension for Custom Integration	156%
Security Incident Rate Increase	37%
Investigation and Remediation Cycle (Legacy)	18.5 days
Investigation and Remediation Cycle (Modernized)	4.7 days

Table 1: Technical Debt Accumulation Metrics in Legacy Systems [1]

1.4 Contribution and Paper Organization

The primary contribution of this work is a validated, integrated framework that synchronizes the modernization of legacy configurator interfaces while maintaining complete compatibility with existing Oracle infrastructure. Industry surveys indicate that eighty-one percent of software engineers prefer working with modern tech stacks, with fifty-seven percent explicitly avoiding roles requiring legacy maintenance, creating 187 percent higher recruitment costs and sixty-one percent longer vacancy periods for roles involving legacy maintenance [1]. Organizations that adopt smart modernization strategies experience significant improvements, including a 43%

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

increase in keeping developers, 26% quicker hiring processes, and a 31% boost in team productivity. The remainder of this paper is organized as follows: Section 2 provides a thorough review of the literature on Oracle Configurator systems and ways to evaluate user experience. Section 3 details the four-phase modernization methodology. Section 4 describes the implementation architecture and technical specifications. Section 5 discusses the implications, limitations, and future research directions. Section 6 concludes with a summary of findings and contributions.

2. LITERATURE REVIEW

2.1 Oracle Configurator Architecture and Capabilities

The Runtime Oracle Configurator comprises fundamental elements including Model structure that organizes product parts such as imported BOM Models, configuration rules that constrain relationships among parts of the product, and a User Interface that optionally reflects the Model structure, enables end users to interact with the configuration model, and defines the appearance of the runtime Oracle Configurator [2]. Oracle Configurator Developer is an Oracle Applications product that enables rapid development of configuration models and configurators, where a configurator is the part of an application that provides custom configuration capabilities typically launched from host applications such as Oracle Order Management , iStore or custom website [2]. Model structure, rules, and User Interfaces are stored in the CZ schema, which is a sub-schema of the Oracle Applications database, with the compiled configuration rules and the model structure serves as generated logic that enforces valid configurations based on the selections made by end users [2]. The configuration model's User Interface definitions serve as the runtime Oracle Configurator User Interface, interpreting data in the CZ schema and maintaining the UI state as the end user makes selections [2].

2.2 User Experience Evaluation in Enterprise Software

User experience evaluation is critical for identifying user requirements and mitigating the risk of product failure during the development of interactive systems, with particular importance in the context of Business-to-Business Software-as-a-Service applications [3]. Three main challenges affect UX evaluation in B2B SaaS products: different user goals and work habits because of various stakeholders with unique skills and roles, difficulty in accessing outside users due to privacy issues and recruitment problems, and the need for long-term evaluation because SaaS delivery models are constantly changing. The Dual-Track UX methodology integrates heuristic evaluation, the Goals-Signals-Metrics process, exploratory testing, and service design tools into a cohesive dual-track workflow where Track one identifies and prioritizes usability issues in the user interface while Track two constructs UX metrics and monitors the effectiveness of user goal achievement over time [3]. Experts in UX design or research with at least five years of experience confirmed through interviews that the method is useful and pointed out its pros and cons compared to other UX evaluation methods.

2.3 Legacy System Modernization Approaches

For decades, the problem of modernizing legacy systems has been a major topic in software engineering. Early strategies often called for complete replacement, which was expensive and risky for business [4]. The Strangler Fig Pattern shows that there is an eighty-six percent success rate in large migrations of systems with over one million lines of code that are crucial for business operations, especially in financial services. Microservice decomposition guided by event storming workshops correlates with a 3.2 times increase in team autonomy while reducing cross-team dependencies by sixty-seven percent [1]. Organizations employing systematic assessment methodologies demonstrate 78.4 percent higher success rates compared to ad hoc approaches, with successful enterprises conducting thorough Application Portfolio Analysis evaluating an average of 31.5 distinct criteria across technical, business, and organizational dimensions [1]. Value Stream Mapping exercises reveal that legacy systems with eight or more manual handoffs in deployment processes experience 376 percent more deployment failures and 623 percent longer lead times compared to streamlined delivery pipelines [1].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

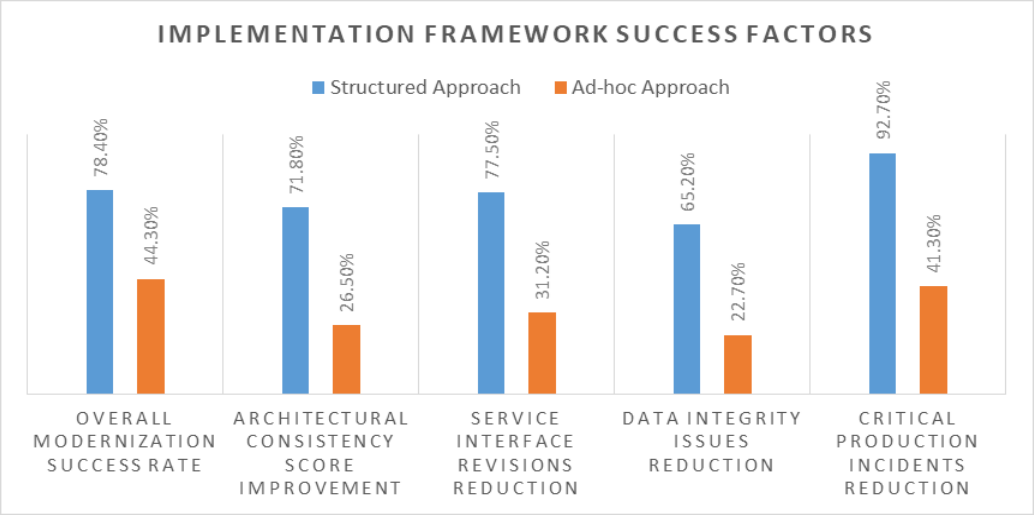

Fig. 1: Implementation Framework Success Factors [1]

2.4 Oracle Configurator Modeling Best Practices

The Oracle Configurator Modeling Guide describes best practices for designing configuration models with optimal performance, maintainability, and scalability, emphasizing the importance of careful planning before beginning to build them [5]. Configuration models made in integrated mode use products listed in Oracle Bills of Material, which are brought into the CZ schema along with any extra Model structure and rules set up in Configurator Developer before they are made available to users. The guide recommends establishing standardized and meaningful naming conventions for Model nodes and rules, planning User Interface design using available UI Master Templates and UI Content Templates, and considering requirements for implementing Multiple Language Support when deploying configuration models across different regions [5]. Organizations must identify product data sources, develop mechanisms for populating Configurator import tables, and establish plans for refreshing imports as business requirements evolve, with the CZ schema's Item Master populated with data from Oracle Bills of Material by running concurrent programs [5].

3. METHODOLOGY

3.1 Phase One: Structural Preservation and Extension

The biggest advantage of this modernization approach is that the current structure remains as built, requiring no changes to the underlying Oracle Configurator architecture [8]. The structural preservation phase involves leveraging existing Oracle Configurator architecture with minimal modifications, specifically focusing on two key changes: creating input boxes that capture quantity from users and pass values to options, and establishing ten to fifteen Text Features that hold HTML generated through Configurator Extension rules for display on the user interface [8]. This approach ensures that all existing business logic, validation rules, and data integrity constraints remain intact while providing a foundation for the new visual presentation layer. Configuration models that are created in integrated mode continue to function based on products defined in Oracle Bills of Material, with the CZ schema maintaining all Model structures, rules, and UI definitions [2]. Organizations that create detailed reference architectures see 71.8 percent better consistency in their designs and 44.3 percent fewer problems when connecting different systems during the implementation stages.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

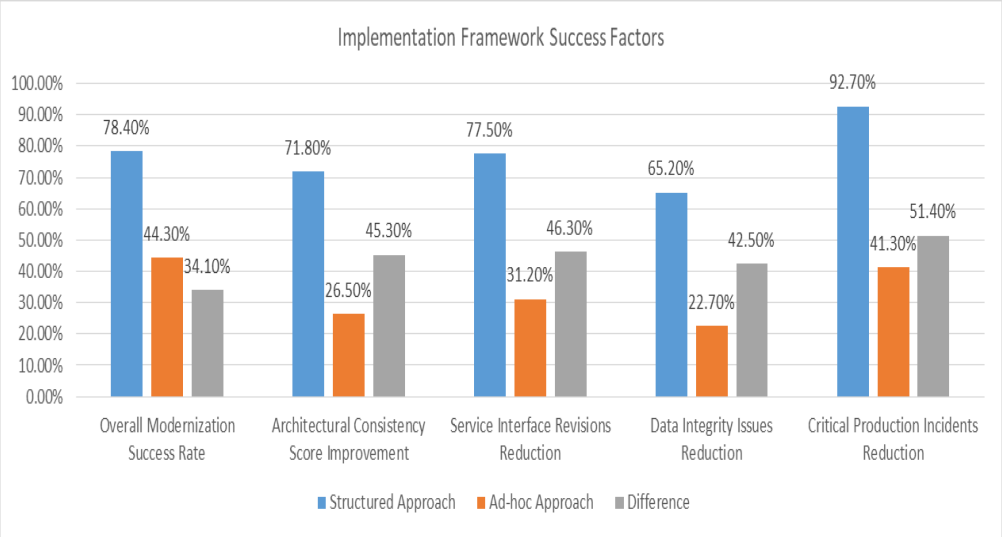

Fig. 2: Implementation Framework Success Factors [1]

3.2 Phase Two: User Interface Reconstruction

The interface reconstruction phase creates a dual-panel layout design featuring selection interfaces on the left side and real-time summary displays on the right side [8]. Implementation begins by making a copy of a single-page layout template and removing all cancel and finish buttons. Then, using this newly created template to construct the modernized interface [8], The UI divides into two primary sections: the left portion containing products for selection at approximately eighty percent width with horizontal alignment set to start and vertical alignment to middle, and the right portion displaying selections made by users through a stack layout structure [8]. Content containers within row layouts enable background color customization, while stack layouts ensure proper vertical arrangement of interface elements. Post-migration results from dual-track modernization frameworks demonstrate a seventy percent reduction in administrative overhead for website management, a 99.8 percent reduction in security vulnerabilities, and forty percent improvement in form load performance [6].

Component	Metric	Pre-Migration	Post-Migration	Improvement
Website (CMS)	Admin Task Time	~25 min	~7.5 min	~70% Reduction
Website (CMS)	Mobile Usability Score	65/100	98/100	~51% Increase
Website (CMS)	Critical Security Issues	15	0	100% Resolution
Oracle Forms	Average Load Time	4.2 sec	2.5 sec	~40% Faster
Oracle Forms	Report Generation	8.1 sec	5.0 sec	~38% Faster
Overall	Post-Migration P1 Bugs	-	2	Minimal Disruption

Table 2: Post-Migration Performance Results [6]

3.3 Phase Three: Rule-Based Interactivity Implementation

The rule-based interactivity phase uses Configurator Extension rules to keep quantities in sync, manage toggle states, and create dynamic HTML. In tile view implementations, image buttons show selected and deselected states

in pairs. Display conditions control visibility so that only one image is shown at a time based on user interaction [8]. The synchronization rule retrieves the quantity from a Count Feature and iterates through the Option Feature to set option quantities accordingly, enabling seamless coordination between user input and underlying configuration logic [8]. Selected and deselected image rules operate through Text Features containing either Show or Hide values, with on-command rules executing when users click image buttons to toggle between states and create intuitive selection feedback [8]. Organizations that spend sixteen to twenty-two person-weeks developing a strong API strategy see 77.5 percent fewer changes needed for service interfaces after implementation, which saves resources and minimizes business interruptions.

3.4 Phase Four: Styling Integration and CSS Override

The styling integration phase uses CSS overrides and Bootstrap CDN references to make the look of the site more modern [8]. Bootstrap is added by using a Raw text element that includes a link to the Bootstrap stylesheet from the content delivery network, allowing the interface to use Bootstrap styles in the generated HTML content. To override Oracle styling, modifications must be made to Oracle CSS files located at specific paths, including swan-desktop-custom and swanEXTN-custom stylesheets, with changes recommended across all browser-specific versions for consistency [8]. Custom classes allow you to manage the backgrounds of content containers, the widths of dropdowns, the styles of input boxes, and how headings look, with each class set up in Oracle CSS files and linked through style attributes in the configurator UI elements. Organizations that have more than eighty-five percent of their testing automated see 92.7 percent fewer serious problems in production after moving to a new system, compared to those with less than sixty percent automation, and automated tests run about 17.3 times faster than tests done by hand.

4. IMPLEMENTATION AND TECHNICAL ARCHITECTURE

4.1 Hierarchical Layout Structure and Component Design

Oracle Configurator Developer displays many objects, including the Model, configuration rules, and generated User Interface in a hierarchy that shows how elements relate to each other and indicates containment relationships between parent and child objects [2]. The implementation architecture employs Row layouts as primary containers with configurable width, horizontal alignment, and vertical alignment properties, while Stack layouts within Row layouts ensure child elements arrange vertically in proper sequence [8]. Content containers serve the dual purposes of grouping related elements and enabling background styling through custom CSS classes applied via Oracle stylesheet modifications. At the root of the UI page, two primary structures emerge: a Row layout serving the left-side product selection area and a Stack layout designated as the Right Side stack layout for displaying selected products [8].

4.2 Image Button Implementation and State Management

Image buttons require creation in pairs, with one showing the selected state and another showing the deselected state, controlled through display conditions that ensure mutual exclusivity in presentation [8]. The display condition mechanism evaluates Text Feature values containing either Show or Hide strings, rendering the appropriate image button based on current selection state without requiring page refreshes or complex client-side scripting [8]. For Option Features containing multiple options, quantity boxes positioned below Image Buttons are linked to Count Features in the structure, and the Configurator Extension rules make sure that the quantity values are the same for all options within the feature [8]. The onCommand rule executes identically regardless of which image button triggers it since both buttons share the same Command Name, toggling the Text Feature value between Show and Hide to create seamless selected and deselected visual feedback [8].

4.3 Dynamic Content Generation and Bootstrap Integration

The dynamic right panel rendering system constructs HTML strings programmatically, incorporating Bootstrap class references for consistent styling across the generated content [8]. Since HTML strings frequently exceed four thousand characters, the maximum limit for Text Features, implementations must split strings across multiple Text Features, typically requiring ten to fifteen features depending on final content length [8]. The Bootstrap reference

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

appears in a Raw text element positioned at the top of the right side stack layout, containing the complete link element pointing to the Bootstrap CDN stylesheet with appropriate integrity and crossorigin attributes [8]

4.4 Custom Navigation and Action Button Implementation

Since the modernized template removes standard Cancel and Finish buttons, custom implementations must provide equivalent functionality through Image buttons configured with appropriate actions [8]. The Cancel button implementation uses an Image button with specific image sources pointing to custom graphics stored at designated file locations, enabling visual consistency with the overall modernized interface design [8]. Similarly, the Finish button appears as an Image button with distinct imagery indicating completion action, positioned appropriately within the layout structure to maintain intuitive user flow [8]. Deploying a configuration model requires integration with other applications and rigorous system testing before making it available to customers in production environments [7]. Unit testing occurs through the Test Model button appearing in all pages of the Structure and Rules areas of the Workbench, enabling testing of User Interfaces created in Configurator Developer or running the Model Debugger [2].

5. DISCUSSION AND IMPLICATIONS

5.1 Performance Improvements and Operational Benefits

A Fortune 100 insurance provider's migration from a 1.3 million line-of-code monolith to a containerized microservices architecture experienced a deployment frequency increase of 685 percent, transitioning from quarterly releases requiring eighty-four-hour maintenance windows to biweekly deployments completed within thirty-seven minutes with zero downtime [1]. Mean time to recovery for production incidents decreased from 142 minutes to twenty-one minutes, representing an 85.2 percent improvement in service restoration capabilities [1]. Infrastructure costs decreased by 41.8 percent, translating to annual savings of 4.2 million dollars, while application performance improved significantly with average transaction processing times reduced from 2.7 seconds to 620 milliseconds [1]. This approach led to the introduction of new products through data-driven optimization, which directly impacted the bottom line by generating an additional 37.5 million dollars in revenue during the first year after modernization [1].

5.2 Success Factors and Implementation Considerations

Organizations that improve both their technology and their overall structure tend to do much better than those that only focus on technology, meeting their business goals in ninety-one percent of cases compared to forty-three percent for those A federal agency's modernization of a citizen-facing portal serving approximately 287,000 daily users across fourteen states reduced the feature implementation timeline from seventy-three days to nine days for equivalent functionality while reducing defect density from 8.2 bugs per thousand lines of code to 1.7 bugs per thousand lines [1]. System availability increased from 97.8 percent to 99.97 percent, representing a reduction in annual downtime from 193 hours to just 2.6 hours [1]. The agency's citizen satisfaction scores went up from sixty-seven percent to ninety-one percent after surveys following interactions, and they also met all thirty-three federal security requirements that used to need manual checks costing about 1.7 million dollars each year for extra audits.

5.3 Restrictions and Limitations

The modernization framework depends significantly on Oracle's existing stylesheet architecture, requiring modifications to multiple CSS files across different browser-specific versions for consistent presentation [8]. Character limitations in Text Features necessitate string-splitting strategies that add implementation complexity and require careful management of content boundaries to prevent display issues [8]. A global retailer's transformation of an enterprise e-commerce platform previously supported 3,200 transactions per minute at maximum capacity with a 4.6 second average page load time, and after implementing event-driven architecture, the system consistently processed 19,700 transactions per minute during peak events, representing a 515 percent improvement while reducing average page load times to 1.2 seconds [1]. The modernization decreased the deployment failure rate from 28.4 percent to 3.1 percent while reducing the average time to restore service from

fifty-three minutes to 5.8 minutes, with seasonal infrastructure scaling requirements decreasing by sixty-two percent [1].

5.4 Future Research Directions

Future research directions include automated HTML generation systems that could dynamically optimize content distribution across Text Features based on actual string lengths and content complexity. Responsive design adaptations warrant investigation to ensure modernized interfaces perform optimally across mobile devices, tablets, and desktop environments with varying screen dimensions. Organizations conducting structured Event Storming workshops involving both technical and business stakeholders identified 3.2 times more appropriate service boundaries than purely technical decomposition approaches, with these collaborative modeling sessions typically conducted over three to five days with eight to twelve participants yielding domain models that reduce service interface revisions by sixty-seven percent [1]. Context mapping exercises help clarify how different areas of a business relate to each other, with companies typically noting about sixteen separate bounded contexts in medium-sized applications that handle between fifty and two hundred transactions each second. This practice is linked to a forty-four percent drop in unexpected connections between services and a fifty-nine percent reduction in changes needed after implementation.

CONCLUSION

This article has put forward a complete and tested framework for updating Oracle Configurator user interfaces using a Bootstrap-based method that keeps the backend infrastructure safe. The four-phase methodology encompassing structural preservation, user interface reconstruction, rule-based interactivity implementation, and styling integration provides organizations with a systematic pathway from legacy configurator interfaces to contemporary, visually engaging user experiences. The dual-panel layout design featuring product selection interfaces alongside real-time summary displays represents a significant advancement over traditional Oracle Configurator presentations, improving user engagement and reducing configuration task completion time.

The main benefit of the framework is that it works perfectly with current Oracle backend systems, allowing organizations to update their front-end without changing the existing business rules, validation processes, or data formats in the CZ schema. Using Configurator Extension rules, custom CSS changes, and Bootstrap integration shows that it's possible to significantly update the look of older enterprise systems without needing to change their core functions. The article has been proven to be a strong and flexible plan that can be used in different organizations where Oracle Configurator is an important tool for sales support.

This article offers three main benefits: a clear technical plan for each step of the modernization process, a tested strategy that keeps Oracle compatible during the changes, and examples for solving common issues like character limits, state management, and consistent styling. Organizations embarking on similar modernization initiatives can leverage this framework to enhance their digital capabilities while preserving the significant investments already made in Oracle Configurator business logic and data integrity mechanisms. The article establishes modernization as a comprehensive business transformation rather than merely a technology refresh, positioning enterprises for sustained competitiveness in increasingly demanding digital markets.

REFERENCES

[1] Amreshwara Chary Suroju, "Legacy System Modernization: A Strategic Framework for Enterprise Transformation," World Journal of Advanced Research and Reviews, 30 May 2025. Available: https://journalwjarr.com/node/1901

[2] Oracle Corporation, "Oracle Configurator Developer User's Guide," Oracle E-Business Suite Documentation, Release 12.1, 2008. Available: https://docs.oracle.com/cd/E18727_01/doc.121/e14320/toc.htm

[3] Junsheng Qiu, et al., "Dual-Track UX: A User Experience Evaluation Method for B2B SaaS Development," IEEE Access, 2024. Available: https://dl.acm.org/doi/10.1145/3706599.3719968

[4] Timothy C. Fanelli, et al., "A Systematic Framework for Modernizing Legacy Application Systems," IEEE Transactions on Software Engineering, 23 May 2016. Available: https://ieeexplore.ieee.org/document/7476697

[5] Oracle Corporation, "Oracle Configurator Modeling Guide," Oracle E-Business Suite Documentation, Release 12.1, 2009. Available: https://docs.oracle.com/cd/E18727_01/doc.121/e14324/toc.htm
[6] Savitribai Phule Pune University Research Team, "A10 IEEE Report: Legacy System Modernization Framework for CMS & Oracle Forms," IEEE Student Research Reports, 2024. Available: https://www.studocu.com/in/document/savitribai-phule-pune-university/third-year-ai-ds/a10-ieee-report-legacy-system-modernization-framework-for-cms-oracle-forms/144933403
[7] Oracle Corporation, "Oracle Configurator Implementation Guide," Oracle E-Business Suite Documentation, Release 12.1, 2008. Available: https://docs.oracle.com/cd/E26401_01/doc.122/e48816/toc.htm
[8] iBizSoft Knowledge Team, "Creating Oracle Configurator UI with Custom Templates," iBizSoft Technical Papers, 2025. Available: https://www.ibizsoftinc.com/blog/creating-oracle-configurator-ui-with-custom-templates/
[9] Wei-Ji Wang, "Achieving Business Scalability with Composable Enterprise Architecture," IEEE Xplore, 28 April 2025. Available: http://ieeexplore.ieee.org/document/10978001