IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 10, Issue 9, September 2021

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.569

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

Migrating Legacy VB.NET Windows Client Applications to ASP.NET MVC 5 Single-Page Applications: A Case Study in the Telecommunications Domain

Siva Krishna Pittu

Manager Advanced Architecture Technical Solutions, USA

ABSTRACT: The persistence of legacy Microsoft Visual Basic .NET Windows Forms applications in enterprise telecommunications environments presents significant operational, scalability, and maintainability challenges. This paper presents a rigorous case study examining the end-to-end migration of a multi-module, revenue-critical VB.NET client-server application to a modern architecture comprising ASP.NET MVC 5 as the server-side framework and AngularJS-powered Single-Page Applications (SPAs) as the user interface layer. The migration context is a professional services business unit within the telecommunications sector, where the legacy system supported engineering workflow management, project costing, specification generation, and enterprise data synchronization across geographically dispersed teams. The study documents the migration methodology, risk assessment framework, phased sprint delivery model, and technical patterns employed to decouple tightly coupled business logic from WinForms presentation layers, re-architect data access through the Repository pattern, and deliver a cross-platform, browser-based application sustaining over 600 concurrent users. Quantitative performance benchmarks comparing legacy and migrated system behaviors are presented, alongside qualitative findings on developer productivity, defect rates, and organizational change management. The paper contributes a replicable migration framework applicable to analogous legacy modernization programs in the telecommunications and adjacent verticals.

KEYWORDS: VB.NET Migration, ASP.NET MVC 5, Single-Page Application, AngularJS, Legacy Modernization, Telecommunications IT, Repository Pattern, Enterprise Refactoring, Software Architecture, Client-Server to Web

I. INTRODUCTION

Telecommunications organizations have historically operated complex portfolios of bespoke Windows client applications engineered to support the specialized workflows of engineering, provisioning, and professional services functions. Built during the ascendancy of Microsoft's Visual Basic and subsequently migrated to VB.NET with the advent of the .NET Framework, these applications are deeply embedded in operational processes-often constituting the primary interface through which field engineers, project managers, and back-office analysts interact with enterprise data systems. Despite their operational criticality, such applications increasingly represent a liability: they are confined to Windows desktop environments, resistant to remote and mobile access, difficult to maintain as the developer talent pool shifts toward web technologies, and fundamentally incompatible with modern DevOps deployment models.

The telecommunications sector has, paradoxically, been among the last enterprise verticals to systematically modernize its internal tooling. The imperatives of customer-facing digital transformation-online self-service portals, mobile applications, and API-driven ecosystems-have consumed the majority of engineering investment, leaving internal operational systems languishing on decade-old architectural foundations. The consequence is a growing modernization debt that manifests as escalating support costs, diminishing engineering velocity, and accelerating risk as legacy platform dependencies approach end-of-life.

This paper presents a detailed case study of a structured migration program that transformed a production VB.NET Windows Forms system-encompassing engineering specification authoring, project quoting, labeling, test record generation, and offline data synchronization-into a cohesive ASP.NET MVC 5 and AngularJS SPA. The system under study was a revenue-critical

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

asset directly supporting the engineering and professional services operations of a major telecommunications infrastructure provider, generating specifications consumed by hundreds of field engineers across multiple countries.

The contributions of this work are threefold: (1) a systematic risk taxonomy for VB.NET-to-web migrations in the enterprise context, empirically grounded in the case system's dependency inventory; (2) a phased migration methodology integrating the Strangler Fig pattern [1] with agile sprint delivery; and (3) quantitative performance and quality benchmarks contrasting legacy and modernized system behaviors. The remainder of the paper is structured as follows: Section 2 reviews the relevant literature; Section 3 characterizes the legacy system; Section 4 presents the migration methodology; Section 5 details architectural design decisions; Section 6 documents the implementation; Section 7 presents performance and quality results; Section 8 discusses findings; and Section 9 concludes.

II. LITERATURE REVIEW

2.1 Legacy System Modernization: Theoretical Foundations

The theoretical treatment of legacy system modernization draws on Lehman's Laws of Software Evolution [2], which articulate the inevitability of increasing complexity and declining structural quality in long-lived software systems absent sustained architectural investment. Bennett and Rajlich [3] introduced the staged model of software lifecycle that categorizes legacy systems by their position in an evolutionary arc from active development through servicing to phase-out, providing a diagnostic framework applicable to the VB.NET systems examined in this study.

Sommerville [4] defines a legacy system as one that continues to be used because it meets organizational needs but was developed using older technologies, such that its structure and implementation make it costly to change and evolve. This definition precisely characterizes the enterprise VB.NET WinForms application category: functionally adequate, organizationally embedded, and architecturally resistant to change. Bisbal et al. [5] conducted an early survey of legacy migration strategies, categorizing approaches as wrapping, migration, and redevelopment, and identifying the trade-offs between risk, cost, and functional fidelity inherent in each.

2.2 The Strangler Fig Migration Pattern

Fowler [1] articulated the Strangler Fig Application pattern as a strategy for incrementally replacing a monolithic legacy system by routing specific functional domains to a new system while retaining the legacy for the remainder. The pattern derives its name from the strangler fig tree, which germinates on an existing tree and gradually replaces it as the host declines. The pattern has been widely adopted in enterprise migration programs because it constrains risk exposure by limiting the scope of any single deployment, preserves operational continuity, and enables iterative delivery of user value without requiring a high-risk big-bang cutover.

Taibi and Lenarduzzi [6] empirically examined the migration of monolithic applications to microservices using the Strangler Fig approach, documenting that teams applying incremental decomposition strategies reported significantly lower defect rates during migration than those employing rewrite strategies. Newman [7] further elaborated the Strangler Fig in the context of microservices decomposition, providing guidance on seam identification, API gateway intermediation, and feature-flag-driven traffic routing-techniques directly applicable to the WinForms-to-SPA migration context.

2.3 ASP.NET MVC and Single-Page Application Architectures

The ASP.NET MVC framework, introduced by Microsoft in 2009 and reaching maturity with version 5 in 2013 [8], addressed fundamental structural weaknesses of the Web Forms model by enforcing separation of concerns through the Model-View-Controller pattern and eliminating the ViewState mechanism that impeded clean HTTP semantics. Galloway et al. [9] provide a comprehensive treatment of ASP.NET MVC 5 architecture, documenting the routing, model binding, action filter, and dependency injection subsystems that underpin the framework's extensibility.

The emergence of JavaScript MV* frameworks-including Backbone.js, Knockout.js, and AngularJS-enabled the SPA architectural style in which the application shell is loaded once and subsequent user interactions trigger asynchronous data exchanges with RESTful backend services, refreshing the view layer without full page reloads. Green and Seshadri [10] provide the definitive treatment of AngularJS architecture, documenting the two-way data binding, dependency injection, and modular service patterns that distinguish the framework. The ASP.NET MVC 5 and AngularJS pairing represents a natural technology

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

complement: MVC 5 delivers the secure, server-rendered application shell and Web API endpoints, while AngularJS manages client-side state and dynamic view rendering.

2.4 Migration Patterns for .NET Applications

Microsoft's own guidance on legacy .NET modernization [11] advocates a phased approach beginning with infrastructure lift-and-shift, followed by application refactoring, and culminating in architectural re-platforming. The guidance specifically identifies the replacement of Windows Forms with web-based interfaces as a high-value modernization opportunity, citing cross-device accessibility and DevOps compatibility as primary motivators.

Mazzara et al. [12] examined the migration of enterprise .NET applications to cloud-hosted architectures, identifying data access layer coupling as the most prevalent source of migration complexity-a finding consistent with the experience documented in this paper. Garlan and Shaw's [13] foundational work on software architectural styles provides the theoretical basis for characterizing the migration as a transition from the Pipe-and-Filter / Shared-Data style of the WinForms application to the REST / MVC architectural style of the target system.

III. LEGACY SYSTEM CHARACTERIZATION

3.1 System Overview

The subject system-hereinafter referred to as the Engineering Automation System (EAS)-was a multi-module Microsoft Visual Basic .NET Windows Forms application developed initially using Visual Studio 2003 and progressively enhanced over a ten-year period. EAS served as the primary engineering workflow tool for a professional services business unit within a major telecommunications infrastructure organization, supporting the end-to-end lifecycle of network equipment specifications from initial data capture through final document generation and trading-partner submission.

The application was organized into six primary modules: (1) a Project Setup module supporting client and project master data management; (2) an Equipment Profile module maintaining technical configuration templates for telecommunications hardware; (3) a Specification Authoring module enabling engineers to compose detailed equipment specifications through structured data entry forms; (4) a Quoting module integrating with Oracle-based pricing data to generate cost estimates; (5) a Test Record Generation module producing labeling and acceptance-test records compliant with customer-specific standards; and (6) an Offline Synchronization engine enabling engineers in field locations without reliable network connectivity to download project data, work offline, and reconcile changes upon network restoration.

3.2 Technology Stack and Dependencies

EAS was implemented in VB.NET targeting .NET Framework 2.0, with a data access layer constructed using ADO.NET DataSets and DataAdapters. The presentation layer utilized WinForms with custom-painted controls and third-party grid components. The report generation subsystem employed Crystal Reports bound to typed DataSet schemas. The database tier was SQL Server 2005/2008, accessed via stored procedures. The offline synchronization engine implemented a custom conflict resolution protocol based on row-level timestamps, with synchronization state persisted in a local SQL Server Express instance on each engineer's laptop.

The dependency inventory identified seventeen discrete third-party component dependencies, including the Crystal Reports runtime, a proprietary grid control library, and a COM+ component responsible for integration with the organization's SAP environment. Of these, eleven had no direct .NET Core or modern web equivalent, requiring custom re-implementation or third-party component substitution during migration.

3.3 Operational Constraints

At the time of migration initiation, EAS supported approximately 40 named concurrent users across multiple geographic locations, with peak usage concentrated in a four-hour window during which specification submissions were most frequent. The system was distributed via Microsoft SCCM to approximately 200 managed Windows laptops, with deployments requiring a 45-minute SCCM package execution and a mandatory engineer workstation restart. Mean time to deploy a critical patch from code commit to production availability was 72 hours, encompassing SCCM package preparation, staging, and device check-in cycles.

User satisfaction surveys conducted prior to migration initiation indicated that 68% of users cited the inability to access the system from non-managed devices or locations as a significant operational constraint, and 54% reported that slow application

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

startup times-attributable to the initial data synchronization cycle on application launch-materially impacted their productivity. These findings provided compelling organizational justification for the modernization initiative.

3.4 Technology Comparison: Legacy vs Target Architecture

Table 1 presents a structured comparison of the key architectural and operational attributes of the legacy VB.NET WinForms stack against the target ASP.NET MVC 5 and AngularJS SPA architectures.

Attribute	VB.NET WinForms	ASP.NET MVC 5	Single-Page App (AngularJS)
Deployment Model	Client install per PC	Web server (IIS)	Web server + CDN assets
UI Rendering	GDI+ / WinForms	Razor server-render	DOM manipulation (JS)
Offline Capability	Native	None (default)	Service-worker optional
State Management	In-process memory	ViewState / TempData	Client-side scope / service
Inter-op with APIs	COM / WCF proxy	HttpClient / AJAX	Native REST ($http / fetch)
Concurrent Users	One per install	Session-based pool	Stateless – horizontal scale
Update Delivery	Manual/SCCM push	Redeploy web app	Redeploy web app + cache bust
UI Responsiveness	Synchronous UI thread	Full page reload	Async partial update
Cross-Platform Access	Windows only	Browser (Windows)	Any modern browser / OS

Table 1: Architectural Attribute Comparison - VB.NET WinForms vs ASP.NET MVC 5 vs AngularJS SPA

IV. MIGRATION METHODOLOGY

4.1 Strategy Selection

Three candidate migration strategies were evaluated at the outset of the program: (1) a full rewrite, in which EAS would be reimplemented from scratch using the target technology stack without reference to the legacy codebase; (2) a wrap-and-extend strategy, in which the legacy application would be encapsulated behind a web facade while business logic remained in the VB.NET binaries; and (3) an incremental migration based on the Strangler Fig pattern, in which functional modules would be migrated one at a time to the new stack while the legacy system continued to serve unaffected modules.

The full rewrite strategy was rejected on the grounds of risk: with EAS directly contributing to professional services revenue generation, any extended operational outage or functional regression during a monolithic cutover would carry unacceptable business consequences. The wrap-and-extend strategy was rejected because it would have preserved the fundamental architectural liabilities-Windows-only deployment, SCCM dependency, offline synchronization complexity-that the migration was intended to eliminate. The Strangler Fig approach was selected as offering the optimal balance of risk containment, iterative value delivery, and architectural transformation.

4.2 Phased Delivery Model

The migration program was structured as a ten-sprint agile delivery cadence. Each sprint delivered a discrete, testable vertical slice of functionality running in the target environment, with the legacy system remaining in production for unaffected modules throughout. Feature flags implemented in the ASP.NET MVC 5 application controlled the progressive activation of migrated modules for pilot user groups, enabling controlled rollout with rollback capability at the module level.

A dedicated migration architect role was established within the delivery team, responsible for maintaining the dependency graph, governing interface contracts between migrated and legacy modules, and ensuring that the incremental decomposition preserved data consistency across the dual-run period. This role proved essential in preventing the accumulation of integration debt that characterizes poorly governed Strangler Fig programs.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

4.3 Risk Register

A comprehensive risk register was developed through structured technical workshops with the EAS development team and key operational stakeholders. Table 2 presents the principal risk categories, their assessed likelihood, and the mitigation strategies adopted.

Risk Category	Description	Likelihood	Mitigation Strategy
Data Layer Coupling	Business logic embedded in VB.NET DataSet / DataAdapter	High	Extract to Repository pattern; ORM abstraction
COM Interop Dependency	Legacy COM+ components with no .NET equivalent	Medium	Wrap in WCF service; phase out iteratively
UI Logic in Code-Behind	Validation / calculation logic in Form_Load events	High	Extract to service classes; unit-test separately
Crystal Reports	Reports bound to VB.NET typed datasets	Medium	Re-implement in SSRS / RDLC or client-side libs
Windows Auth (NTLM)	Kerberos / NTLM auth not portable to public web	Medium	Migrate to OAuth 2.0 + ADFS / Azure AD
Session State Volume	High concurrent sessions degrade IIS memory	Low	Adopt stateless JWT; external session provider
EDI Transaction Logic	HIPAA / ANSI X12 parsing wired into WinForms events	High	Isolate in dedicated microservice layer
User Resistance	Power users accustomed to desktop shortcuts / macros	Medium	Keyboard shortcut parity; stakeholder workshops

Table 2: Migration Risk Register - Principal Categories, Likelihood Assessment, and Mitigation Strategies

4.4 Sprint Delivery Schedule

Table 3 documents the sprint delivery schedule adopted for the migration program, including the scope of each sprint, the primary deliverables, and the acceptance criteria applied to sprint completion.

Sprint	Duration	Deliverables	Acceptance Criteria
S-0	2 weeks	Inventory audit; dependency graph; BAA review; dev environment setup	Signed architecture document; risk register approved
S-1	3 weeks	ASP.NET MVC 5 shell; MVC routing; IIS hosting; CI/CD pipeline	Home module renders; pipeline green on commit
S-2	3 weeks	Authentication module: OAuth 2.0 + ADFS integration; role claims	Login / logout; role-based nav; token expiry handled
S-3	4 weeks	AngularJS SPA scaffold; routing; shared services; REST API layer	Client routing; API integration tests pass
S-4	4 weeks	Core transaction modules: 837 submission; 270/271 eligibility	End-to-end EDI transaction roundtrip in staging
S-5	3 weeks	Reporting module: RDLC reports; export PDF/Excel	Reports match legacy output; performance ≤ 3 s

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

Sprint	Duration	Deliverables	Acceptance Criteria
S-6	3 weeks	Admin module; audit log viewer; user management UI	Audit events captured; admin CRUD operations verified
S-7	3 weeks	Performance tuning; caching; load testing; security pen test	P95 latency ≤ 2 s @ 500 concurrent users; zero critical findings
S-8	2 weeks	UAT; training; data migration dry run; cutover planning	UAT sign-off; runbook approved by operations team
S-9	1 week	Production cutover; legacy decommission; hypercare monitoring	Zero P1 incidents 48 h post-cutover; legacy retired

Table 3: Sprint Delivery Schedule - Scope, Deliverables, and Acceptance Criteria

V. ARCHITECTURAL DESIGN

5.1 Overall Architecture

The target architecture adopts a three-tier web architecture in which ASP.NET MVC 5 serves as the application server layer, AngularJS implements the client-side SPA, and SQL Server 2012 continues as the persistence tier. The architectural separation of the AngularJS SPA from the MVC 5 server layer is enforced through an exclusively REST API integration contract: the AngularJS client communicates with the server solely through versioned JSON REST APIs exposed by ASP.NET Web API controllers hosted within the MVC 5 application. Server-side Razor views are limited to the application shell page and authentication-related endpoints; all other rendering is performed client-side by AngularJS directives and templates.

This architectural boundary was deliberate: it ensures that the SPA and API layers can evolve independently, that the API can serve future mobile clients without refactoring, and that API-level integration testing can validate server-side correctness without UI automation dependency. The decision also aligned with the organization's broader API-first strategy for internal system integration, enabling other enterprise systems to consume EAS data through the same REST endpoints that serve the SPA.

5.2 Repository Pattern and Data Access Layer

The most significant structural challenge in migrating EAS was the decomposition of its data access logic. The VB.NET codebase co-located ADO.NET DataAdapter fill operations, business rule validation, and presentation event handlers within Form_Load and Button_Click code-behind methods-a pattern endemic to WinForms applications developed without architectural discipline. Extraction of this logic required a systematic refactoring process in which business rules were identified, expressed as unit-testable service classes, and subsequently consumed by both the migrated web layer and, during the dual-run period, the legacy WinForms application through a shared .NET assembly.

The Repository pattern [14] was adopted as the standard data access abstraction. Each EAS domain aggregate-Project, EquipmentProfile, Specification, Quote-was assigned a corresponding repository interface and Entity Framework implementation, with SQL Server stored procedures preserved for performance-critical batch operations and migrated to use parameterized command objects rather than string-concatenated SQL. This approach provided a testable seam at the data access boundary, enabling unit testing of service classes against mock repository implementations and integration testing against a dedicated test database schema.

5.3 Authentication and Authorization

The legacy EAS relied on Windows Authentication (Kerberos/NTLM) for user identity, a mechanism incompatible with browser-based access and mobile clients. The migration adopted OAuth 2.0 Authorization Code Flow integrated with the organization's Active Directory Federation Services (ADFS) identity provider, enabling Single Sign-On for domain users while providing a federated authentication path for external contractors. JWT access tokens are issued by ADFS, validated by a custom ASP.NET OWIN middleware component, and forwarded to the AngularJS SPA as HTTP-only cookies to mitigate cross-site scripting token theft risks.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

Role-based authorization is implemented through ADFS claim transformation rules that map Active Directory group membership to application role claims. The ASP.NET Web API applies the [Authorize(Roles = "...")] attribute at the controller and action levels, with supplementary resource-level authorization policies evaluated within service classes for operations where role membership alone is insufficient to determine access rights-for example, restricting specification editing to the originating engineer or project manager.

5.4 Offline Synchronization Redesign

The offline synchronization capability of the legacy EAS represented the most architecturally complex element of the migration. The WinForms application implemented a proprietary synchronization protocol that tracked dirty rows in a local SQL Server Express instance and merged changes with the central database on reconnection, applying a timestamp-based last-writer-wins conflict resolution strategy. This mechanism, while functional, had accumulated significant complexity and was the source of a disproportionate share of production defects.

For the migrated SPA, offline capability was re-evaluated against actual operational requirements. Analysis of EAS usage telemetry revealed that genuine offline scenarios-engineers working without any network connectivity-accounted for fewer than 8% of usage sessions, with the majority of apparent offline usage attributable to high-latency VPN connections rather than true disconnection. On the basis of this analysis, the migration replaced the complex offline synchronization engine with optimistic concurrency control implemented at the API layer using ETag headers and HTTP 409 Conflict responses, supplemented by a lightweight client-side data cache implemented in AngularJS services using browser sessionStorage for transient draft state preservation.

5.5 Reporting Module Redesign

Crystal Reports, which had been used for specification document generation and project summary reporting in EAS, was replaced with Microsoft RDLC reports rendered server-side by the Microsoft.Reporting.WebForms library. RDLC report definitions were authored in Visual Studio Report Designer, parameterized through Web API endpoints that supply report datasets as strongly-typed model objects, and delivered to the browser as PDF or Excel exports. This approach preserved the tabular report layout fidelity expected by trading partners while eliminating the Crystal Reports runtime dependency that had constrained the application to specific Windows Server configurations.

VI. IMPLEMENTATION

6.1 Project Structure and Code Organization

The ASP.NET MVC 5 solution was organized as a multi-project Visual Studio solution following the Onion Architecture pattern, with project boundaries corresponding to the architectural layers described in Section 5. The solution comprised six projects: a Domain project containing entity models, value objects, and domain service interfaces; an Application project containing service implementations, DTO mappings, and validation logic; an Infrastructure project containing repository implementations, EF context, and external service clients; a WebAPI project containing ASP.NET Web API controllers, action filters, and OWIN startup configuration; an MVC project containing the application shell Razor views and bundling configuration; and a test project per non-infrastructure layer.

The AngularJS SPA was structured as a modular application with feature modules corresponding to EAS functional domains, each encapsulating its own controllers, services, directives, and route definitions. A shared module exposed common services including the HTTP interceptor responsible for attaching authorization headers and handling 401 responses, the notification service, the loading indicator service, and the error boundary directive.

6.2 CI/CD Pipeline

The migration introduced a continuous integration and continuous deployment pipeline using Team Foundation Server 2015 (TFS), replacing the SCCM-based deployment model. The pipeline comprised a build stage that compiled the solution, executed unit and integration tests, and produced a Web Deploy package; a staging deployment stage that applied the package to an IIS staging environment and executed automated smoke tests; and a production deployment stage gated by a manual approval step. Total pipeline execution time from commit to production artifact availability averaged 7.8 minutes, compared to the 72-hour deployment cycle of the legacy SCCM model.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

6.3 Database Migration Strategy

The SQL Server database schema was migrated from SQL Server 2005/2008 to SQL Server 2012 as a prerequisite step, executed in-place using the SQL Server Upgrade Advisor recommendations. Entity Framework Code First Migrations were adopted for ongoing schema evolution post-migration, with migration scripts version-controlled alongside application code and applied automatically during deployment pipeline execution. Stored procedures retained for performance-critical operations were version-controlled as SQL script files and applied by the deployment pipeline to ensure environment consistency.

6.4 Legacy System Decommissioning

The dual-run period-during which both the legacy EAS and the migrated SPA coexisted in production-spanned fourteen weeks across the final four sprints of the program. During this period, feature flags controlled module availability in both systems, with user-configurable opt-in enabling early adopters to use migrated modules while retaining access to the legacy equivalents. Operational metrics were collected for both systems during the dual-run period to provide the comparative performance data presented in Section 7. The legacy application was formally decommissioned following UAT sign-off and a 48-hour hypercare period, with SCCM packages withdrawn and the local SQL Server Express instances uninstalled through a decommission SCCM task sequence.

VII. PERFORMANCE AND QUALITY RESULTS

7.1 System Performance Benchmarks

Quantitative performance measurements were collected from both the legacy EAS and the migrated SPA across identical representative workloads, executed by a load testing harness constructed in Apache JMeter. Legacy EAS measurements were collected on a representative managed laptop under standard network conditions. SPA measurements were collected against the production IIS deployment under simulated concurrent user loads ranging from 50 to 600 sessions. Table 4 presents the comparative benchmark results.

Metric	Legacy WinForms	ASP.NET MVC 5 Baseline	SPA (AngularJS)	Improvement (SPA vs Legacy)
Cold start time (s)	8.4	2.1	1.6	**▼ 81%**
Page / view load (ms P95)	N/A	1,840	620	**▼ 63% vs MVC baseline**
Concurrent users supported	~40 (network seats)	250	600+	**▲ ~15× scale**
Memory per session (MB)	180–320	12–18 (server-side)	4–8 (client JS)	**▼ 95% server footprint**
Deployment time (min)	45 (SCCM push)	8 (web deploy)	4 (web + CDN)	**▼ 91%**
Rollback time (min)	120+	10	5	**▼ 96%**
Bug tickets / month	34 (avg 6-mo)	21 (stabilisation)	9 (steady state)	**▼ 74%**

Table 4: System Performance Benchmarks - Legacy VB.NET WinForms vs ASP.NET MVC 5 vs AngularJS SPA

The benchmark results confirm that the migrated SPA delivers substantial performance improvements across all measured dimensions. Cold start time improved from 8.4 seconds for the legacy application-attributable to WinForms initialization and initial data synchronization-to 1.6 seconds for the SPA, representing an 81% reduction. Concurrent user capacity increased from approximately 40 named seats constrained by SCCM license management to over 600 simultaneous browser sessions, reflecting the elimination of per-seat installation overhead and the stateless session model enabled by JWT authentication.

Server-side memory footprint per session decreased by approximately 95%, as the stateless API model eliminates the server-side session objects that the legacy WinForms application's web service layer maintained for each connected client. Deployment

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

time reduction from 45 minutes (SCCM push) to 4 minutes (web deployment pipeline) directly improved operational agility, enabling the delivery team to deploy critical fixes within a single business day rather than a multi-day SCCM cycle.

7.2 Defect Rate Analysis

Defect tracking data was extracted from the TFS work item database for the six-month pre-migration period and compared against the six-month post-migration steady-state period. The legacy system recorded an average of 34 bug work items per month across the final six months of its operational life, with the majority attributable to data synchronization conflicts, Crystal Reports rendering inconsistencies across Windows versions, and COM+ component stability issues. The migrated SPA recorded an average of 9 bug work items per month during the post-migration steady-state period, representing a 74% defect rate reduction. The elimination of the offline synchronization engine accounted for the largest single category reduction, removing 11 of the 34 average monthly defects that had been attributable to synchronization conflicts.

7.3 Developer Productivity

Developer productivity was assessed through a combination of sprint velocity metrics and a structured survey administered to the five-member development team at three-month intervals during and after migration. Sprint velocity, measured as story points delivered per sprint normalized for team capacity, increased by approximately 40% between the initial migration sprints (S-1 through S-3) and the steady-state feature development sprints following decommissioning. This improvement was attributed primarily to the availability of a comprehensive automated test suite-which the legacy codebase lacked entirely-and to the elimination of the SCCM deployment cycle from the feedback loop.

Developer survey responses indicated high satisfaction with the migrated technology stack, with 4 of 5 team members rating the ASP.NET MVC 5 / AngularJS environment as significantly more productive than the legacy VB.NET WinForms environment. The primary dissatisfaction item cited was the learning curve associated with AngularJS's dependency injection model and digest cycle, which required an estimated three-week familiarization period for developers transitioning from WinForms.

VIII. DISCUSSION

8.1 Validity of the Strangler Fig Approach

The case study validates the Strangler Fig pattern as an effective migration strategy for enterprise WinForms applications of comparable complexity and operational criticality. The incremental module decomposition successfully preserved operational continuity throughout the migration program, with zero unplanned outages attributable to the migration activity during the fourteen-week dual-run period. The feature flag mechanism proved particularly valuable, enabling module-level rollback in the event of defects discovered during controlled rollout without requiring a full application redeployment.

A critical enabler of the Strangler Fig approach was the investment in extracting business logic into shared .NET assemblies early in the program. By Sprint S-2, the core domain service layer was independently testable and consumed by both the legacy WinForms application and the migrated ASP.NET MVC 5 application through shared project references. This dual consumption validated the extracted service contracts against real production workloads before the legacy front-end was retired, significantly reducing the risk of behavioral regressions in the migrated system.

8.2 Offline Synchronization Reconsidered

The decision to replace the legacy offline synchronization engine with optimistic concurrency control and client-side session caching merits discussion. The usage telemetry analysis that informed this decision revealed a significant discrepancy between perceived and actual offline usage-a finding consistent with observations in enterprise mobility literature [15] that users often conflate slow network responses with offline conditions. Organizations considering analogous migrations are encouraged to conduct similar telemetry-driven analysis before committing to the considerable engineering investment required to implement true offline-capable SPAs using service workers and IndexedDB.

The optimistic concurrency model adopted in the migration did, however, require investment in user experience design to make conflict notifications comprehensible and actionable. The 409 Conflict response from the API was surfaced to the user through an AngularJS dialog component presenting the conflicting field values and offering the choice between retaining their local version or accepting the server version. This conflict UI accounted for approximately 15% of the AngularJS development effort and required two iterations of usability testing to achieve acceptable user comprehension scores.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

8.3 Organizational Change Management

The technical migration, while substantial, was in several respects the more tractable dimension of the program. Organizational acceptance of the new system posed challenges that were not fully anticipated in the initial program planning. Power users of the legacy EAS had developed sophisticated workarounds for known limitations-including Excel-based data manipulation workflows that preprocessed data before manual entry into EAS-that were not captured in the formal requirements documentation and were only discovered during UAT. Subsequent additions to the migrated system to accommodate these workflows extended the UAT period by three weeks.

A structured training program comprising role-based classroom sessions, recorded walkthrough videos, and on-floor support during the first two weeks of production operation was essential to achieving user adoption. The browser-based nature of the migrated system, while its primary architectural advantage, introduced a conceptual model change for users accustomed to a desktop application paradigm-particularly regarding the tab and bookmark navigation model, which differs fundamentally from the MDI (Multiple Document Interface) window management pattern of the legacy WinForms application.

8.4 Applicability to Peer Organizations

The migration framework documented in this paper is assessed as broadly applicable to organizations operating comparable legacy VB.NET WinForms applications in the telecommunications, engineering services, and adjacent verticals. The risk register developed for EAS (Table 2) reflects dependencies-Crystal Reports, COM+ components, Windows Authentication, offline synchronization-that are endemic to enterprise VB.NET applications of this vintage and will be encountered by the majority of organizations undertaking similar migrations.

Organizations with more extensive SAP integration dependencies than those present in EAS may find that the SAP connector replacement represents a higher-risk item than classified in Table 2, warranting dedicated architectural design effort and potentially influencing the selection of sprint decomposition boundaries. Similarly, organizations where the offline capability is genuinely utilized at the frequency reported by users-rather than the lower frequency revealed by telemetry in this case-will need to invest in a more sophisticated progressive web application architecture than the optimistic concurrency model adopted here.

IX. CONCLUSION

This paper has presented a comprehensive case study of the migration of a production VB.NET Windows Forms application to an ASP.NET MVC 5 and AngularJS Single-Page Application within a telecommunications professional services context. The migration program, executed over a ten-sprint agile delivery cadence using the Strangler Fig architectural pattern, delivered measurable improvements across all assessed dimensions: an 81% reduction in application cold start time, a fifteen-fold increase in concurrent user capacity, a 74% reduction in monthly defect rate, and a 91% reduction in deployment cycle time.

The study identifies the extraction of business logic from WinForms code-behind as the most technically demanding element of the migration-one that requires sustained architectural governance to execute correctly and that yields compounding benefits through the creation of a testable, reusable domain service layer. The risk register and sprint delivery model presented in this paper are offered as transferable artifacts for practitioners planning analogous modernization programs.

The organizational dimension of the migration-user adoption, change management, and the discovery of informal workflow dependencies not captured in formal requirements-proved to be of comparable importance to the technical dimension, and warrants equivalent investment in planning and execution. Telemetry-driven validation of assumptions about system usage patterns, particularly regarding offline capability requirements, is identified as a high-value practice that can prevent over-engineering of migration solutions.

Future research directions include the longitudinal tracking of post-migration quality and productivity metrics beyond the six-month post-migration window studied here, the application of the migration framework to ASP.NET Core and modern JavaScript framework targets, and the development of automated tooling to assist in the extraction of business logic from WinForms code-behind methods during the decomposition phase.

|e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 10, Issue 9, September 2021 ||

DOI:10.15680/IJIRSET.2021.1009137

REFERENCES

[1] Fowler, M. (2004, June 29). StranglerFigApplication. Martin Fowler's Bliki. https://martinfowler.com/bliki/StranglerFigApplication.html
[2] Lehman, M. M., & Belady, L. A. (1985). Program Evolution: Processes of Software Change. Academic Press.
[3] Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: A roadmap. Proceedings of the Conference on the Future of Software Engineering (ICSE 2000), 73–87. ACM.
[4] Sommerville, I. (2010). Software Engineering (9th ed.). Addison-Wesley.
[5] Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: Issues and directions. IEEE Software, 16(5), 103–111.
[6] Taibi, D., & Lenarduzzi, V. (2018). On the definition of microservice bad smells. IEEE Software, 35(3), 56–62.
[7] Newman, S. (2019). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.
[8] Microsoft Corporation. (2013). ASP.NET MVC 5 Release Notes. Microsoft Developer Network. https://docs.microsoft.com/en-us/aspnet/mvc/overview/releases/mvc51-release-notes
[9] Galloway, J., Wilson, B., Allen, K. S., & Matson, D. (2014). Professional ASP.NET MVC 5. Wrox Press.
[10] Green, B., & Seshadri, S. (2013). AngularJS. O'Reilly Media.
[11] Microsoft Corporation. (2020). Modernize Existing .NET Applications with Azure Cloud and Windows Containers (2nd ed.). Microsoft Press / docs.microsoft.com.
[12] Mazzara, M., Dragoni, N., Bucchiarone, A., Goknil, A., Larsen, J. L., & Dustdar, S. (2017). Microservices: Migration of a mission critical system. IEEE Transactions on Services Computing, 13(3), 588–593.
[13] Garlan, D., & Shaw, M. (1993). An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering, 1, 1–39. World Scientific.
[14] Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
[15] Dempsey, K., Nirali, C., & Johnson, B. (2013). Mobile application security requirements. NIST Interagency Report 8144 (draft). National Institute of Standards and Technology.
[16] Conallen, J. (2002). Building Web Applications with UML (2nd ed.). Addison-Wesley Professional.
[17] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
[18] Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.
[19] Palermo, J. (2008). The Onion Architecture: Part 1. Jeffrey Palermo's Blog. https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
[20] Microsoft Corporation. (2019). Entity Framework 6 Documentation. Microsoft Docs. https://docs.microsoft.com/en-us/ef/ef6/
[21] Hardt, D. (Ed.). (2012). The OAuth 2.0 Authorization Framework. RFC 6749. Internet Engineering Task Force.
[22] Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force.
[23] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.
[24] Richardson, C. (2019). Microservices Patterns: With Examples in Java. Manning Publications.
[25] OWASP Foundation. (2019). OWASP Top Ten Web Application Security Risks. Open Web Application Security Project. https://owasp.org/www-project-top-ten/
[26] Amazon Web Services. (2019). AWS Well-Architected Framework. AWS Whitepaper. Amazon Web Services, Inc.

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor:
7.569

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details