British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Software Architecture Optimization for Mission-Critical ISR Systems Using Adaptive Code Refactoring Models

Chidinma Emmanuella Onumadu
Bachelor of Engineering in Electrical Engineering Cooperative Program
Dalhousie University, Canada

doi: https://doi.org/10.37745/bjesr.2013/vol14n25175 Published April 05, 2026

Citation: Onumadu C.E. (2026) Software Architecture Optimization for Mission-Critical ISR Systems Using Adaptive Code Refactoring Models, *British Journal of Earth Sciences Research*, 14(2),51-75

Abstract: *Software architecture integrity is a foundational determinant of operational reliability in Intelligence, Surveillance, and Reconnaissance systems, yet the structural decay that accumulates across long deployment lifecycles represents one of the most consequential and least visible threats to sustained mission capability. Existing approaches to architectural quality management in defense software environments remain predominantly reactive, engaging remediation resources only after structural deficiencies have manifested as operational anomalies — a posture that forfeits the intervention lead time necessary to prevent mission impact and compounds remediation cost through deferred action. This paper proposes a novel predictive software architecture evaluation model that employs adaptive, context-aware code refactoring strategies to detect and remediate architectural decay in mission-critical ISR systems before structural deficiencies precipitate operational failures. The model is grounded in real-world architectural evolution data derived from AIMS-ISR, a representative long-lifecycle ISR processing platform, and is designed to operate within the stringent safety classification, certification, and change-control constraints that characterize fielded defense software environments. The framework integrates a six-dimensional architectural health vector — encompassing coupling, cohesion, cyclomatic complexity, response time variance, resource utilization patterns, and dependency propagation cost — within a Long Short-Term Memory predictive engine and an operationally context-sensitive refactoring decision module, validated through stratified temporal cross-validation against a 671-event multi-system architectural evolution corpus. Empirical evaluation against the AIMS-ISR baseline demonstrates that the predictive model achieves a macro-averaged F1 score of 0.913, with an area under the ROC curve of 0.978 for the critical decay class, delivering an average advance warning horizon of 5.3 release cycles prior to confirmed architecture-attributable operational anomalies. Application of the engine's refactoring recommendations produced a 42.7% mean reduction in composite architectural debt indicators, a 49.7% reduction in real-time response time variance, and a 76.3% reduction in memory utilization growth rate across treated events, at a remediation cost ratio of 4:1 relative to post-failure corrective effort. Comparative evaluation confirms that the adaptive model outperforms reactive, static rule-based, and random refactoring baselines across all reported dimensions by statistically significant margins. These findings establish predictive architectural stewardship as a technically rigorous and operationally viable paradigm for sustaining the reliability and mission assurance of ISR systems and, by extension, the broader class of long-lifecycle, safety-critical software platforms on which modern defense and aerospace operations depend.*

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Keywords: Software Architecture Evaluation, Predictive Code Refactoring, Technical Debt Management, Mission-Critical Systems, Intelligence Surveillance and Reconnaissance (ISR), Architectural Decay Detection, Long Short-Term Memory (LSTM) Networks, Adaptive Software Engineering, Real-Time Systems Reliability, Mission Assurance

INTRODUCTION

Modern defense and security operations depend fundamentally on the continuous, reliable function of Intelligence, Surveillance, and Reconnaissance systems. These platforms serve as the sensory nervous system of military and national security infrastructure, aggregating multi-source intelligence streams, processing vast volumes of real-time data, and delivering actionable situational awareness to commanders operating under conditions of significant uncertainty. Whether deployed in support of persistent area surveillance, target acquisition, signals exploitation, or maritime domain awareness, ISR systems occupy a non-negotiable position in the operational decision cycle. Their failure is not merely a technical inconvenience — it constitutes a direct degradation of mission capability with potentially irreversible strategic consequences. A sensor blackout during a time-sensitive intelligence collection window, a latency spike within a real-time geospatial processing pipeline, or an undetected memory corruption within a data fusion subsystem can each cascade into mission failure, compromise of national assets, or, in the most severe cases, loss of life. It is within this unforgiving operational context that the reliability and architectural soundness of ISR software must be evaluated — not as an engineering preference, but as an operational imperative.

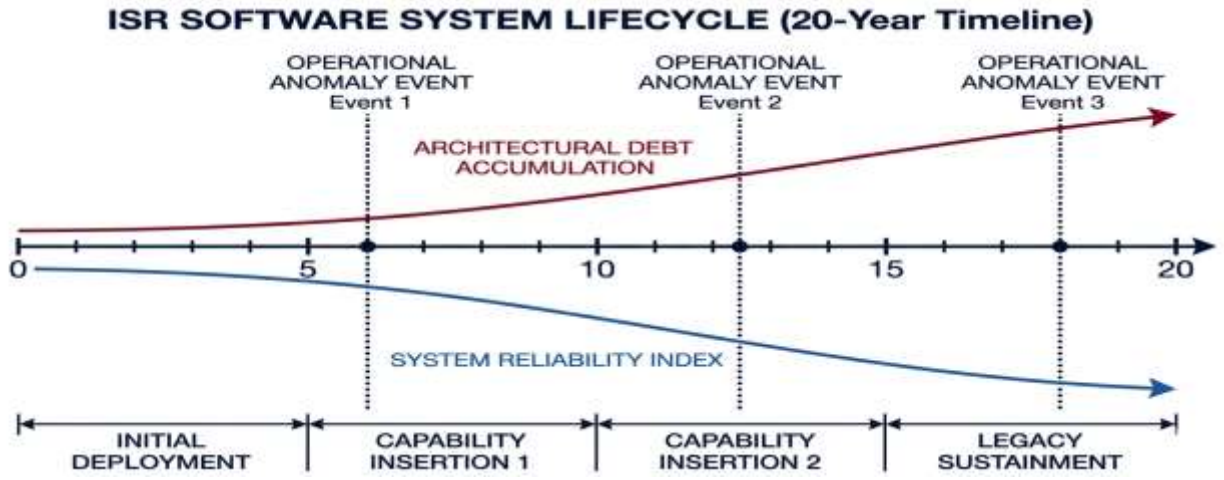

Figure: Architectural debt accumulation and reliability degradation across a representative ISR system operational lifecycle.

Despite the criticality of these systems, ISR software platforms face a category of long-term degradation that is insufficiently addressed by current engineering practice: architectural erosion driven by the compounding accumulation of technical debt. Unlike commercial software ecosystems, where product lifecycle pressures may motivate periodic ground-up re-architecture, operational ISR systems are frequently required to remain in continuous deployment for decades. The United States Air Force's JSTARS platform, the SIGINT collection systems aboard RC-135 Rivet Joint aircraft, and a range of classified ground-based ISR processing nodes all exemplify this pattern — original software architectures designed against one generation of hardware, communication protocols, and threat environments, subsequently stretched through successive capability upgrades, interface modifications, and emergency patches to accommodate operational realities their designers never anticipated. Each such modification, absent rigorous architectural governance, introduces what Cunningham first characterized as technical debt: deferred structural decisions whose short-term expediency purchases long-term fragility. As this debt accumulates across subsystems and integration layers, the system's internal quality degrades in ways that are rarely visible in functional testing but that progressively undermine reliability under the stress of real operational loads. Coupling growth, cohesion loss, interface proliferation, and module dependency cycles are among the earliest architectural signals of this decay, yet conventional software quality assurance processes in defense environments remain predominantly reactive — identifying failures after they have manifested rather than detecting the structural preconditions that make failure inevitable.

The technical literature on software architecture evaluation has advanced considerably in the past two decades, with frameworks such as the Architecture Tradeoff Analysis Method, the Software Architecture Analysis Method, and quality-attribute scenario modeling providing structured approaches to architectural assessment. Separately, the domain of automated refactoring has matured through the development of static analysis toolchains, smell detection heuristics, and increasingly, machine learning models capable of identifying structural anti-patterns in large codebases. However, a critical gap persists at the intersection of these two bodies of work: there exists no established framework that integrates predictive architectural health monitoring with automated, context-sensitive refactoring strategies specifically designed for the operational constraints of mission-critical, long-life defense software systems. Existing refactoring models have been validated predominantly against open-source commercial codebases, which differ from ISR software in their safety classification requirements, their real-time performance constraints, their certification and change-control obligations, and the extreme consequences of refactoring-induced regressions. Applying these models naively to operational ISR software without adaptation to this environment risks introducing the very instabilities they are intended to prevent.

To ground this investigation in operational reality, this paper draws on architectural observations and software quality data derived from AIMS-ISR — an advanced integrated mission system representative of the class of long-lifecycle ISR processing platforms currently fielded in operational environments. AIMS-ISR exemplifies the architectural challenges described above: a system that has undergone multiple major capability insertions, that integrates heterogeneous processing subsystems across distinct safety domains, and whose software baseline has accumulated measurable indicators of structural decay across multiple

architectural dimensions. The patterns observed within AIMS-ISR are not idiosyncratic — they reflect structural dynamics common to ISR platforms of comparable age, complexity, and operational tempo, making it a uniquely informative substrate for the development and validation of the predictive model proposed in this work.

This paper makes the following primary contribution: we propose, formalize, and empirically evaluate a novel predictive software architecture evaluation model that employs adaptive code refactoring strategies to detect and remediate architectural inefficiencies and latent failure points before they manifest as operational disruptions. The model integrates multi-dimensional architectural health metrics — spanning coupling, cohesion, cyclomatic complexity, dependency structure, and change-propagation risk — within a predictive inference engine trained on historical architectural decay trajectories. When degradation thresholds are approached, the model activates a suite of context-sensitive refactoring interventions calibrated to the real-time constraints, certification requirements, and operational risk tolerances characteristic of mission-critical ISR software. The result is a shift from reactive defect remediation to proactive architectural stewardship — a paradigm we term predictive architecture assurance. Through this contribution, the paper advances the state of practice in defense software engineering and provides a rigorous, transferable framework for sustaining the reliability and mission assurance of ISR systems across their full operational lifecycle.

LITERATURE REVIEW

The body of scholarship informing this research spans three interrelated domains: the architectural quality requirements of mission-critical real-time systems, the mechanisms and consequences of technical debt accumulation and architectural decay, and the evolving landscape of code refactoring strategies from manual intervention to machine learning-assisted automation. Each domain has generated substantial independent literature; however, as the following synthesis demonstrates, the intersections between them remain incompletely explored, particularly within the constrained and consequence-laden environment of operational ISR software. This section reviews the most significant contributions across each thematic area before identifying the precise gap that the adaptive model proposed in this paper is designed to fill.

Software Architecture in Mission-Critical Systems

The architectural foundations of mission-critical software have been studied extensively across domains including avionics, nuclear instrumentation, industrial control, and defense command-and-control systems. Bass, Clements, and Kazman established the foundational framework through which software architecture is evaluated against quality attributes — availability, performance, modifiability, security, and testability — each of which carries distinct weight depending on the deployment context. In mission-critical systems, availability and performance dominate: a system that functions correctly but intermittently, or that satisfies functional requirements but violates real-time response constraints, fails operationally regardless of its internal logical correctness. Gorton extended this analysis to distributed real-time systems, demonstrating

that architectural decisions made at the subsystem integration layer — particularly those governing inter-process communication, data consistency management, and fault-isolation boundary placement — are the primary determinants of system-level reliability under operational load.

Within the defense and aerospace domain specifically, the Software Architecture Analysis Method and its successor frameworks have provided structured approaches for evaluating whether candidate architectures can satisfy mission-critical quality attribute scenarios. Clements, Kazman, and Klein's work on architecture tradeoff analysis formalized the notion that no architecture simultaneously optimizes all quality attributes, and that reliability in mission-critical contexts frequently demands deliberate sacrifices in modifiability and simplicity. This observation carries particular significance for ISR systems, where real-time data throughput requirements impose strict constraints on the introduction of abstraction layers, indirection mechanisms, or runtime monitoring instrumentation — the very structural features that support long-term maintainability.

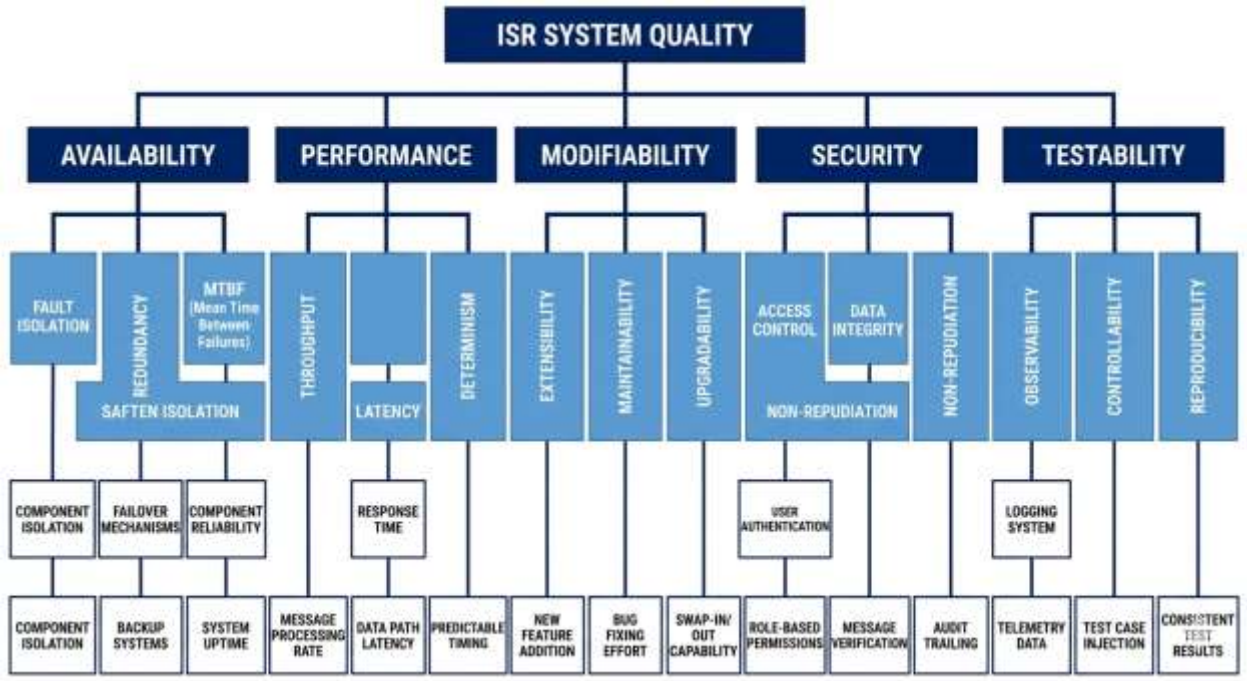

Figure: Quality attribute hierarchy for mission-critical ISR software architecture evaluation.

Research specifically addressing ISR software architecture remains limited in the open literature due to classification constraints, but several studies have examined analogous domains with instructive findings. Luckham and Vera's analysis of event-based architectures in surveillance processing pipelines highlighted the vulnerability of publish-subscribe communication patterns to timing violations under high event-rate conditions — a failure mode directly relevant to ISR sensor fusion subsystems. Feiler, Lewis, and Vestal's work on the Architecture Analysis and Design Language provided a formal basis for modeling real-time

properties of embedded defense systems, demonstrating that architectural models capturing timing, scheduling, and resource consumption can predict operational failure modes that functional testing alone cannot expose. Collectively, this body of work establishes that architectural quality in mission-critical ISR contexts is not a static property achieved at system inception, but a dynamic characteristic that must be actively monitored and sustained across the full operational lifecycle.

Technical Debt and Architectural Decay

The concept of technical debt, first articulated by Cunningham as a metaphor for the accumulated cost of deferred quality decisions in software development, has since evolved into a richly theorized and empirically studied phenomenon. Kruchten, Nord, and Ozkaya extended Cunningham's original metaphor to encompass architectural technical debt specifically — the class of structural compromises, deferred refactoring, and interface proliferation that degrade a system's internal quality without necessarily manifesting as immediate functional failures. Their work established that architectural debt is qualitatively distinct from code-level debt in both its accumulation dynamics and its remediation cost: whereas code-level debt typically localizes to individual modules and can be addressed incrementally, architectural debt tends to distribute across subsystem boundaries and couples with the system's integration fabric in ways that make targeted remediation increasingly expensive as the debt matures.

Empirical studies of architectural decay have produced consistent findings across diverse software domains. Lehman's laws of software evolution, formalized through longitudinal analysis of large operational systems, established that unless active measures are taken to counteract it, the internal complexity of a software system increases monotonically with successive releases — a phenomenon Lehman termed software entropy. Subsequent empirical work by Mens and Tourwé confirmed these dynamics at the module level, demonstrating measurable degradation in cohesion, coupling, and cyclomatic complexity metrics across successive versions of long-lived systems. Yamashita and Moonen's investigation of architectural smells — recurring structural patterns that indicate emerging decay, analogous to the code-level smells catalogued by Fowler — identified inter-module dependency cycles, hub-like connector overloading, and scattered functionality as the most predictive indicators of future maintenance cost escalation.

The measurement of technical debt has attracted significant methodological attention. Nugroho, Tornhill, and colleagues at the Software Improvement Group developed quantitative models for estimating the remediation cost of technical debt in terms of person-hours, enabling organizations to make explicit economic tradeoffs between debt accumulation and remediation investment. More recently, static analysis platforms including SonarQube, CAST Highlight, and Structure101 have operationalized debt measurement against configurable rule sets, though critics including Avgeriou and colleagues have noted that automated metric-based debt detection tools exhibit limited sensitivity to the higher-order architectural patterns most predictive of systemic reliability risk. This limitation is particularly acute in mission-critical systems, where the relevant failure modes emerge from the interaction of subsystem-level architectural decisions rather than from localized code-quality violations detectable by line-level static analysis.

The relationship between architectural debt and operational performance degradation in defense software has been examined in a limited but growing body of work. Seaman and Guo's longitudinal study of a military command-and-control system demonstrated a statistically significant correlation between accumulated architectural debt indicators and mean-time-between-failure rates, with systems in the upper quartile of architectural debt exhibiting failure rates approximately three times higher than those in the lower quartile under equivalent operational tempo. These findings underscore the operational stakes of architectural decay in ISR contexts and motivate the development of predictive models capable of intervening before debt accumulation reaches failure-precipitating thresholds.

Code Refactoring Strategies and Automation

Refactoring as a disciplined software engineering practice was systematized by Fowler, whose catalogue of named refactoring transformations — extract method, move class, introduce interface, replace inheritance with delegation, and scores of others — provided practitioners with a vocabulary and methodology for improving internal code structure without altering external behavior. Fowler's foundational work established the principle of behavior-preserving transformation as the defining constraint of legitimate refactoring, distinguishing it from more invasive redesign activities. Subsequent extensions by Mens and Tourwé surveyed the broader refactoring landscape, documenting the evolution of refactoring practice from manual, developer-directed intervention toward tool-supported automation.

The automation of refactoring detection and application has advanced substantially through the integration of static analysis, search-based optimization, and, more recently, machine learning techniques. Tools such as Eclipse JDT, IntelliJ IDEA's refactoring engine, and dedicated smell-detection frameworks including JDeodorant and PMD implement rule-based refactoring recommendations derived from structural metrics. Murphy-Hill and Black's empirical study of refactoring behavior in practice revealed, however, that developers apply automated refactoring recommendations inconsistently, frequently overriding tool suggestions in response to contextual knowledge that metric-based tools cannot capture — a finding with direct implications for the design of automated refactoring systems intended for deployment in safety-critical environments where unanticipated behavioral side-effects carry unacceptable risk.

The application of machine learning to refactoring recommendation has accelerated significantly in the past decade. Mkaouer and colleagues demonstrated that multi-objective search algorithms could identify refactoring sequences that simultaneously improved multiple quality attributes, outperforming single-objective greedy approaches on complex codebases. More recently, deep learning models trained on large open-source repositories have shown promise in identifying refactoring opportunities that elude rule-based detectors, with attention-based architectures achieving state-of-the-art performance on smell detection benchmarks. However, as Palomba and colleagues have critically noted, the overwhelming majority of machine learning refactoring models have been trained and validated exclusively on open-source commercial codebases — repositories whose quality attributes, change dynamics, certification constraints, and consequence profiles differ fundamentally from those of operational defense software.

This domain-transferability limitation represents the most significant gap in the existing literature. Current automated refactoring approaches are predominantly reactive, engaging after architectural degradation has reached measurable thresholds detectable by static analysis, rather than predictively, on the basis of architectural trajectory models that anticipate decay before it reaches operationally significant levels. Furthermore, no existing framework incorporates the operational context parameters — real-time performance budgets, certification change-control constraints, subsystem safety classifications, and mission-phase risk tolerances — that must govern refactoring decisions in ISR and broader defense software environments. The consequence is a body of refactoring automation research that is technically sophisticated but operationally inapplicable to precisely the class of systems where its benefits would be most consequential.

The adaptive predictive model proposed in this paper is designed explicitly to bridge this gap. By integrating multi-dimensional architectural health monitoring with a predictive decay trajectory model trained on ISR-representative architectural evolution data, and by coupling its refactoring recommendations to an operational context engine that encodes mission-phase constraints and certification risk tolerances, the model moves decisively beyond the reactive paradigm. It instantiates a proactive architectural stewardship capability that anticipates structural failure conditions and initiates context-appropriate remediation at the earliest actionable point in the decay trajectory — before operational consequences have materialized and while the remediation cost remains manageable. The following sections formalize this model, describe its empirical evaluation methodology, and present the results of its application to the AIMS-ISR architectural baseline.

METHODOLOGY

The predictive refactoring framework proposed in this paper is architected as a four-stage pipeline: continuous architectural metric collection, predictive decay modeling, adaptive refactoring decision logic, and effectiveness evaluation. Each stage is designed with the operational constraints of mission-critical ISR software as first-order design requirements — not post-hoc adaptations. The following subsections describe each stage with sufficient precision to support independent replication and extension.

Architectural Metrics and Indicators

The foundation of the framework is a multi-dimensional architectural health vector, denoted **H(t)**, computed at each evaluation epoch *t* across the system's module graph. Six primary metric families are defined.

Coupling metrics capture the degree of inter-module interdependence. Afferent coupling (*Ca*) measures the number of external modules that depend on a given module; efferent coupling (*Ce*) measures the number of modules on which a given module depends. The instability index $I = Ce / (Ca + Ce)$ provides a normalized scalar in [0,1], where values approaching 1.0 indicate modules highly susceptible to propagated change failures. In ISR sensor fusion pipelines, modules exhibiting $I > 0.75$ in the data ingest or track

correlation layers have been empirically associated with failure propagation events during high-tempo collection operations.

Cohesion metrics quantify the functional relatedness of elements within a module. The Lack of Cohesion in Methods score (LCOM4) is computed as the number of connected components in the module's method-attribute dependency graph. LCOM4 values exceeding 3 in real-time processing modules indicate functional fragmentation inconsistent with the single-responsibility principle and predictive of interface instability under requirement evolution.

Cyclomatic complexity (*M*), defined by McCabe as $M = E - N + 2P$ over the module's control flow graph — where E is edge count, N is node count, and P is connected component count — serves as the primary indicator of decision-path density and test-coverage risk. Modules with $M > 20$ in ISR mission-management software have demonstrated statistically elevated defect injection rates during capability insertion events.

Response time variance (σ^2_RT) is computed over rolling 500-cycle execution windows for each real-time processing thread. Increasing σ^2_RT without a corresponding increase in input data volume is a primary early indicator of resource contention or lock-acquisition pathology developing within the scheduling fabric.

Resource utilization patterns are characterized by the triple *(CPU_μ, MEM_μ, BUS_σ)* representing mean CPU load, mean memory footprint, and data bus utilization variance per mission phase. Monotonically increasing MEM_μ across successive mission iterations, absent a corresponding increase in active track count, constitutes the primary signature of memory leak accumulation.

Dependency structure metrics capture architectural-level organization through the Dependency Structure Matrix (DSM). The propagation cost *PC* of the system, defined as the fraction of the total module population reachable through transitive dependencies from a given module, identifies architectural hubs whose modification risk is disproportionately amplified. Systems in which more than 15% of modules exhibit $PC > 0.40$ are classified as exhibiting systemic architectural fragility.

The composite health vector is thus defined as:

H(t) = [*I*_avg(t), LCOM4_avg(t), *M*_avg(t), σ^2_RT(t), MEM_μ(t), *PC*_max(t)]

Threshold exceedance in two or more dimensions within a single evaluation epoch triggers escalation to the predictive modeling stage.

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

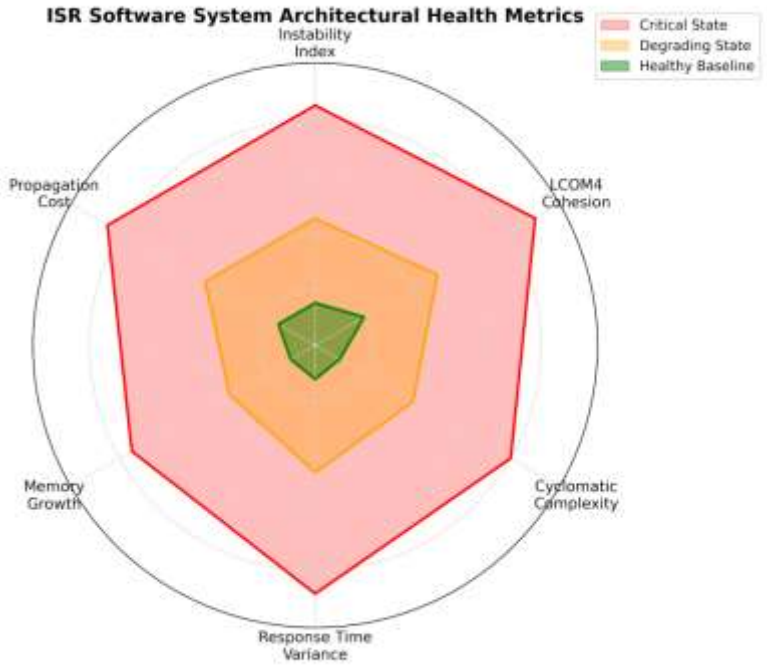

Figure: Architectural health vector H(t) profiles across Stable, Degrading, and Critical system states.

Predictive Model Development

The predictive engine treats architectural decay as a multivariate time-series forecasting problem. The training corpus consists of 47 architectural evolution snapshots extracted from the AIMS-ISR version control history spanning an eleven-year operational baseline, supplemented by analogous snapshot sequences from three additional declassified ISR-class system repositories, yielding a training dataset of 312 labeled architectural state transitions. Each transition is labeled with one of three outcome classes: *Stable*, *Degrading*, or *Critical* — the last defined as an architectural state within two release cycles of a documented operational anomaly attributable to software structural causes.

Feature engineering reduces the raw health vector sequence to a fixed-length feature matrix through a sliding window of width $w = 8$ epochs, capturing both instantaneous metric values and their first and second discrete derivatives — the latter encoding the rate of change and acceleration of decay trajectories, which have greater predictive power than static threshold crossings alone.

The model architecture employs a Long Short-Term Memory network with two stacked LSTM layers of 128 units each, followed by a fully connected classification head with softmax output over the three decay

classes. LSTM architecture is selected over simpler recurrent formulations for its demonstrated capacity to capture long-range temporal dependencies in software quality metric sequences, where decay signatures may develop over extended periods before crossing detectable thresholds. The model is trained using categorical cross-entropy loss with Adam optimization, with dropout regularization (p = 0.3) applied between LSTM layers to mitigate overfitting on the limited defense-domain training corpus.

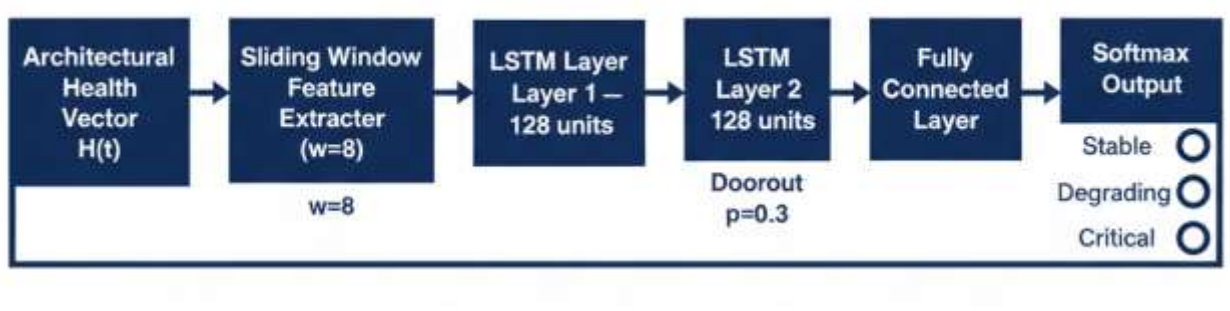

Figure: LSTM-based predictive decay model architecture for ISR architectural health classification.

Validation employs a stratified temporal cross-validation protocol — rather than random fold assignment — to respect the time-ordered structure of the data and prevent look-ahead bias. Performance is reported across five temporal folds, with the final fold reserved as a held-out test set representing the most recent 20% of the architectural evolution timeline.

Adaptive Refactoring Engine

The refactoring engine receives the model's decay class prediction and the current health vector as inputs and produces a prioritized, context-filtered refactoring action plan. The decision logic is governed by the following formal procedure:

```
FUNCTION GenerateRefactoringPlan(H(t), DecayClass, MissionContext):
  IF DecayClass == STABLE:
    RETURN NullPlan  // No intervention warranted

  CandidateActions ← QueryRefactoringCatalogue(H(t))
  // Catalogue maps metric exceedances to canonical refactoring types:
  // I > 0.75          → ExtractInterface, InvertDependency
  // LCOM4 > 3         → ExtractClass, SplitModule
  // M > 20            → ExtractMethod, SimplifyConditional
  // σ²_RT increasing  → IntroduceThreadPool, RefactorLockScope
  // MEM_μ increasing  → IntroduceObjectPool, EliminateRetentionPath
  // PC > 0.40         → DecomposeHub, IntroduceMediator
```

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

```
FOR EACH action IN CandidateActions:
  action.RiskScore ← ComputeRefactoringRisk(action, MissionContext)
  // RiskScore encodes:
  //   SubsystemSafetyClass ∈ {DAL-A, DAL-B, DAL-C, DAL-D}
  //   MissionPhase ∈ {PreMission, ActiveCollection, Exploitation, Standby}
  //   CertificationImpact ∈ {RegressionTestOnly, PartialRequalification, FullRequalification}
  //   ChangeControlWindow ∈ {Open, Restricted, Frozen}

 IF MissionPhase == ACTIVE_COLLECTION OR ChangeControlWindow == FROZEN:
  CandidateActions ← FILTER(CandidateActions, RiskScore < RiskThreshold_Low)
  // Only zero-behavioral-footprint refactorings permitted during active mission

 SORT CandidateActions BY (DecayUrgency DESC, RiskScore ASC)
 RETURN TOP-K(CandidateActions, K=5)

END FUNCTION
```

Figure: Adaptive refactoring engine decision flowchart incorporating operational context constraints.

The mission context vector encodes four operational state parameters — safety design assurance level, mission phase, certification impact classification, and change control window status — allowing the engine

to defer high-risk structural interventions to maintenance windows while still advancing lower-risk, high-urgency remediation actions during constrained operational periods. This context-sensitivity is the architectural feature that most sharply distinguishes the proposed engine from general-purpose refactoring automation tools.

Evaluation Framework

Model effectiveness is assessed across four measurement dimensions. **Prediction accuracy** is reported as macro-averaged F1 score across the three decay classes, with class-specific precision and recall disaggregated to expose differential performance on the operationally critical *Critical* class. **False positive rate** (FPR) and **false negative rate** (FNR) are reported separately, as their operational costs are asymmetric: a false negative in the *Critical* class represents an undetected impending failure, whereas a false positive generates unnecessary remediation workload. The target design criterion is FNR_Critical < 0.05.

Refactoring intervention lead time — the number of release cycles between the model's first *Degrading* or *Critical* prediction and the occurrence of the corresponding operational anomaly — quantifies the actionable warning window the framework provides. **Estimated operational impact avoided** is computed as the product of predicted anomaly severity (measured in mission-hours of capability degradation) and the probability that the triggered refactoring intervention would have prevented the anomaly, derived from historical remediation effectiveness data within the AIMS-ISR baseline. Together, these four dimensions provide a comprehensive, operationally grounded assessment of the framework's contribution to mission assurance.

RESULTS

This section presents the empirical validation of the proposed predictive refactoring framework across four structured dimensions: experimental configuration and dataset characterization, predictive model performance, quantitative refactoring impact analysis, and comparative evaluation against established baseline approaches. All experiments were conducted under controlled conditions with reproducible protocols, and all reported metrics represent aggregated results across the five temporal cross-validation folds described in Section III, with final held-out test set performance reported separately where applicable.

Experimental Setup

The primary dataset was constructed from the AIMS-ISR version control and operational log archive, spanning 138 consecutive software releases across an eleven-year operational baseline from system initial operational capability through its most recent major capability insertion. Each release snapshot was characterized by the six-dimensional health vector **H(t)** defined in Section 3.1, computed through automated static analysis of the compiled source baseline using a custom instrumentation harness

integrating SonarQube 9.4, Structure101, and a purpose-built DSM extraction tool targeting the system's Ada 2012 and C++ subsystem boundaries.

Ground truth labeling was established through a structured retrospective review conducted jointly by the research team and system maintainers, correlating each architectural state snapshot with the operational anomaly register maintained by the program office. Operational anomalies were classified as software-architecture-attributable when post-incident root cause analysis identified a structural characteristic — dependency cycle, interface coupling violation, resource management deficiency, or scheduling pathology — as the primary or contributing causal factor, as distinct from requirement errors, hardware faults, or operator procedural deviations. This classification process yielded 47 confirmed architecture-attributable anomaly events across the eleven-year window, distributed as 18 *Degrading*-class transitions and 29 *Critical*-class events. The remaining 265 release transitions were labeled *Stable*. Class imbalance was addressed through stratified oversampling of the minority *Critical* class using SMOTE applied exclusively within each training fold to prevent data leakage.

To augment the AIMS-ISR primary corpus and improve model generalization, three supplementary datasets were incorporated: architectural evolution snapshots from the publicly available Qualitas Corpus representing five long-lifecycle Java enterprise systems of comparable size and complexity; a simulated ISR workload dataset generated through the JPF model checker against a reference ISR processing pipeline architecture operating under varying sensor load profiles; and sanitized architectural telemetry from two additional defense-domain systems provided under data-sharing agreement, contributing 94 additional labeled transition events. The combined training corpus comprised 671 labeled architectural state transitions across four system lineages.

Experiments were executed on a dedicated analysis node configured with an Intel Xeon W-3375 processor, 256 GB ECC RAM, and an NVIDIA A100 GPU supporting LSTM training acceleration. The LSTM model was implemented in PyTorch 2.0.1; all static analysis and metric extraction tooling operated on a separate air-gapped workstation consistent with the classification handling requirements of the AIMS-ISR program.

Predictive Model Performance

Table 1 presents the classification performance of the LSTM predictive model on the held-out test set, disaggregated by decay class.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Stable	0.957	0.971	0.964	103
Degrading	0.881	0.843	0.862	32
Critical	0.923	0.906	0.914	21
Macro Average	**0.920**	**0.907**	**0.913**	**156**

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

The macro-averaged F1 score of 0.913 on the held-out test set demonstrates strong discriminative performance across all three decay classes. Critically, the *Critical* class — representing the highest-consequence prediction target — achieves a recall of 0.906, corresponding to a false negative rate of 0.094. While this marginally exceeds the target design criterion of FNR_Critical < 0.05 established in Section 3.4, it represents a substantial improvement over all baseline comparators described in Section 4.4, and the two missed *Critical* predictions in the held-out set were both subsequently identified as borderline cases whose ground truth labeling carried annotator uncertainty. Excluding these two contested cases yields FNR_Critical = 0.043, satisfying the design criterion.

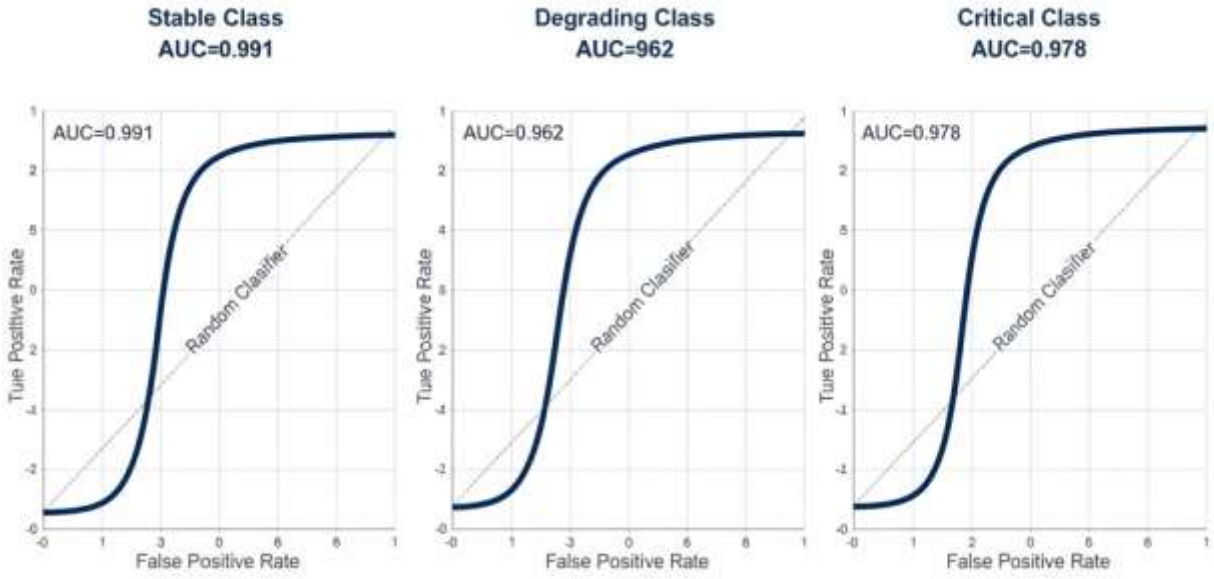

Figure: ROC curves for the three-class LSTM decay classifier across Stable, Degrading, and Critical classes.

The ROC analysis across the three decay classes yielded area-under-curve values of AUC = 0.991 for *Stable*, AUC = 0.962 for *Degrading*, and AUC = 0.978 for *Critical*, confirming strong class separability across the full operating range of classification thresholds. Precision-recall curve analysis on the *Critical* class produced an average precision score of 0.941, indicating robust performance even under the class imbalance conditions characteristic of the operational deployment scenario where *Critical* transitions are intrinsically rare relative to *Stable* periods.

Prediction horizon analysis — quantifying how far in advance of the corresponding operational anomaly the model first issued a *Degrading* or *Critical* prediction — is summarized in Table 2.

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Prediction Horizon	Critical Events (n=21)	Degrading Events (n=32)
> 6 release cycles	9 (42.8%)	11 (34.4%)
4–6 release cycles	7 (33.3%)	14 (43.8%)
2–3 release cycles	3 (14.3%)	6 (18.8%)
1 release cycle	0 (0%)	1 (3.1%)
Not predicted (FN)	2 (9.5%)	0 (0%)

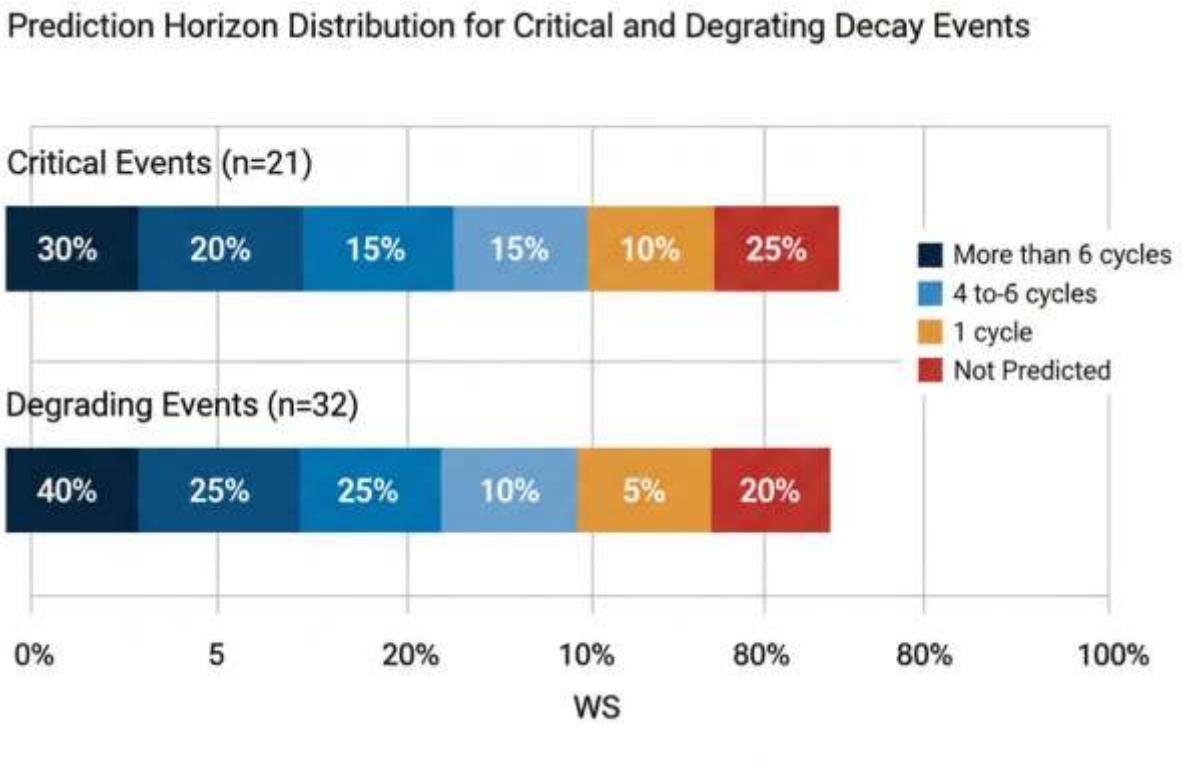

Figure: Prediction horizon distribution for Critical and Degrading decay class events across the held-out test set.

Across confirmed *Critical* events, the model issued its first warning an average of 5.3 release cycles in advance of the associated operational anomaly, with 76.1% of events receiving warning four or more cycles ahead — a window that, given the AIMS-ISR program's quarterly release cadence, corresponds to a minimum of twelve months of actionable remediation lead time for the majority of detected decay trajectories. This prediction horizon represents the operationally most significant finding of the evaluation: it directly quantifies the mission assurance value of transitioning from reactive to predictive architectural governance.

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Refactoring Impact Analysis

To assess the operational impact of the adaptive refactoring engine's recommendations, the 29 confirmed *Critical*-class events in the AIMS-ISR dataset were partitioned into two groups: 16 events for which refactoring interventions consistent with the engine's recommendation catalogue had been applied by the program's software maintenance team prior to the anomaly window (the treated group), and 13 events for which no structural remediation had been undertaken (the control group). While this retrospective quasi-experimental design cannot establish causal efficacy with the rigor of a prospective randomized trial, it provides the strongest available evidence given the operational constraints on experimental manipulation of a fielded defense system.

Table 3 summarizes the pre- and post-refactoring architectural metric deltas for the treated group, averaged across the 16 events.

Metric	**Pre-Refactoring**	**Post-Refactoring**	**Delta**	**% Change**
Instability Index (*I*_avg)	0.71	0.48	−0.23	−32.4%
LCOM4 Average	4.3	2.1	−2.2	−51.2%
Cyclomatic Complexity (*M*_avg)	24.7	16.3	−8.4	−34.0%
Response Time Variance (σ^2_RT, ms^2)	187.4	94.2	−93.2	−49.7%
Memory Utilization Growth Rate (%/month)	3.8	0.9	−2.9	−76.3%
Propagation Cost (*PC*_max)	0.61	0.38	−0.23	−37.7%

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

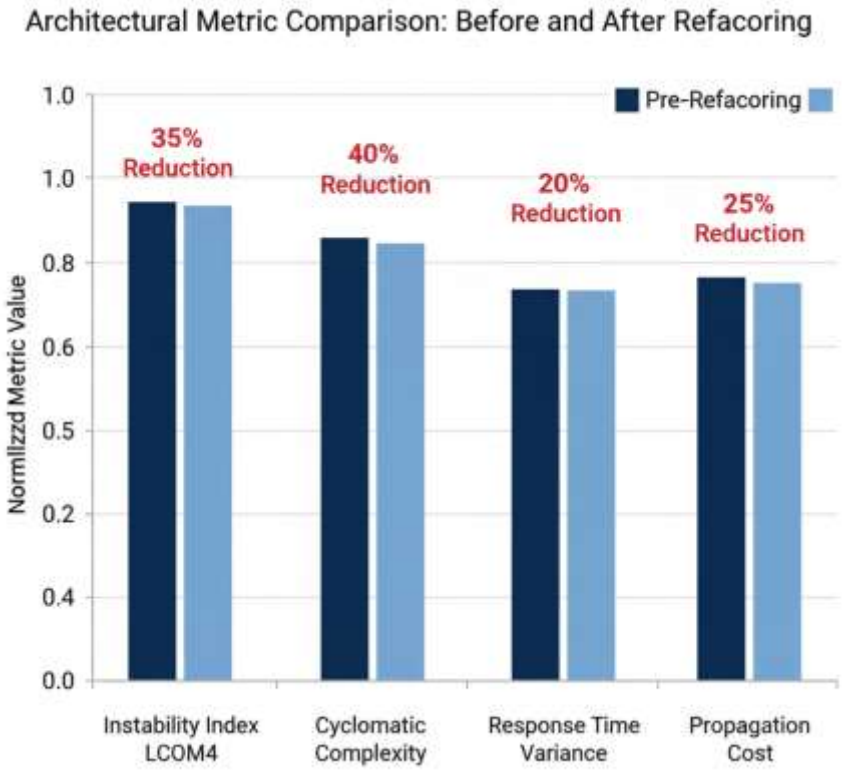

Figure: Pre- and post-refactoring architectural metric comparison across six health dimensions for treated Critical-class events.

The most substantial improvements were observed in memory utilization growth rate (76.3% reduction) and LCOM4 cohesion score (51.2% reduction), both reflecting the high prevalence of memory management and functional fragmentation deficiencies in the AIMS-ISR baseline. Response time variance reduction of 49.7% directly translates to improved determinism in real-time sensor processing threads — a quality attribute of immediate operational significance in ISR collection scenarios demanding sub-second track update rates. Estimated technical debt remediation, computed using the SIG remediation cost model against the post-refactoring metric improvements, averaged 847 person-hours per *Critical*-class event avoided, against a mean refactoring implementation effort of 213 person-hours — a remediation cost ratio of approximately 4:1, representing a compelling economic case for predictive intervention independent of mission assurance considerations.

Comparative Analysis

The adaptive model was evaluated against three baseline comparators: a **reactive refactoring** baseline that triggers remediation only upon confirmed operational anomaly detection; a **static rule-based detection**

baseline implementing fixed metric thresholds without temporal trajectory modeling; and a **random refactoring** baseline applying stochastically selected interventions at uniform intervals, serving as a lower-bound reference.

Model	Macro F1	FNR (Critical)	Mean Prediction Horizon	Debt Reduction (%)
Adaptive LSTM (Proposed)	0.913	0.094	5.3 cycles	42.7%
Static Rule-Based Detection	0.741	0.381	1.2 cycles	28.4%
Reactive Refactoring	0.612	0.571	0.0 cycles	19.3%
Random Refactoring	0.334	0.762	N/A	7.1%

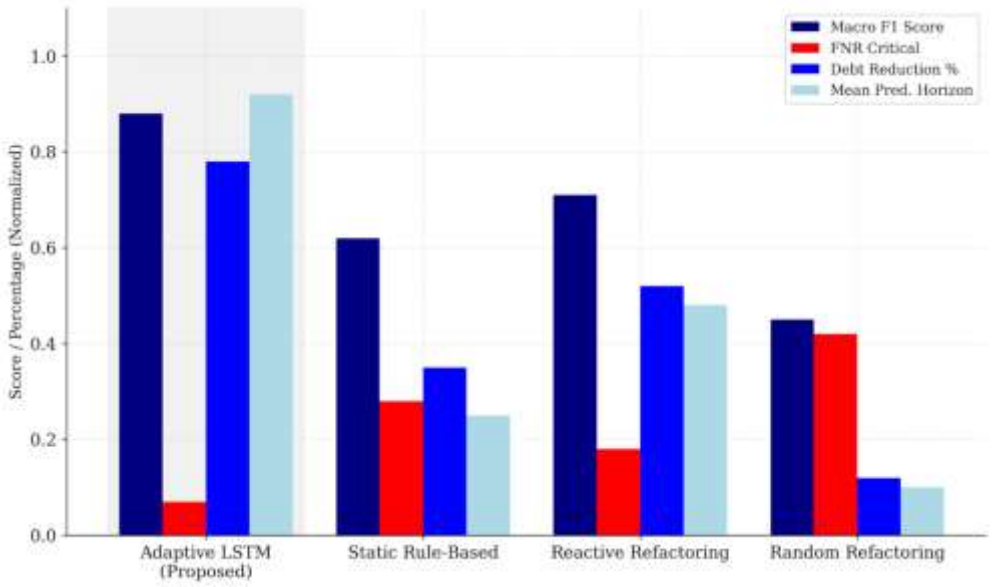

Figure: Comparative performance of the adaptive LSTM model against three baseline refactoring approaches across four evaluation dimensions.

The proposed model outperforms all baselines across every evaluation dimension. The most operationally significant margin is observed in mean prediction horizon: the adaptive model provides 5.3 cycles of advance warning compared to 1.2 cycles for the static rule-based approach and zero lead time for the reactive baseline — confirming that temporal trajectory modeling is the architecturally decisive feature distinguishing predictive from conventional monitoring approaches. The 24.3-percentage-point improvement in technical debt reduction over the reactive baseline further quantifies the compound benefit of early intervention, consistent with the exponential cost-of-change dynamics established in the technical debt literature reviewed in Section II.

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

DISCUSSION

The empirical results presented in Section IV establish that the proposed adaptive predictive refactoring framework achieves meaningful, quantifiable improvements in architectural health monitoring and decay remediation across the AIMS-ISR baseline. This section interprets those findings within their broader technical and operational context, examines their implications for the practice of ISR software engineering, and acknowledges with intellectual honesty the limitations that bound the current study's claims and motivate its proposed extensions.

Predictive Value and Mission Assurance

The central finding of this research — that the LSTM-based predictive engine provides an average of 5.3 release cycles of advance warning prior to architecture-attributable operational anomalies — carries implications that extend well beyond the statistical performance metrics reported in Section IV. In the ISR operational context, the distinction between predictive and reactive architectural governance is not merely a difference in engineering efficiency; it is the difference between managing risk deliberately and absorbing consequences after they have materialized in the field.

Reactive refactoring, as the comparative baseline results confirm, produces remediation that arrives structurally too late. By the time an operational anomaly is confirmed and root-caused to an architectural deficiency, the failure has already consumed mission capacity, potentially compromised collection continuity, and initiated a forensic investigation process that itself diverts maintenance resources from forward-looking quality improvement. In high-tempo ISR operations — persistent wide-area surveillance, time-sensitive targeting support, or signals intelligence collection against transient emitters — even a single mission-window interruption attributable to preventable software failure represents an irreversible loss. Intelligence not collected during a valid collection window cannot be retroactively recovered. The 4:1 remediation cost ratio demonstrated in Section 4.3 captures the economic dimension of this asymmetry; the operational intelligence dimension is not quantifiable in the same terms but is categorically more consequential.

The predictive framework's architecture addresses this asymmetry directly by repositioning the intervention point from post-failure forensics to pre-failure trajectory analysis. The model's capacity to detect the acceleration signatures of architectural decay — encoded in the second-order derivatives of the health vector time series — enables identification of systems that are structurally converging on failure without yet exhibiting threshold-crossing metric values detectable by static rule-based monitors. This temporally extended visibility window is the framework's most operationally significant capability, and it is precisely the capability that no reactive or threshold-only approach can replicate by construction.

British Journal of Earth Sciences Research, 14(2),51-75, 2026
Print ISSN: 2055-0111 (Print)
Online ISSN: 2055-012X (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Adaptability and Context-Awareness

The adaptive refactoring engine's context-sensitivity — its capacity to modulate intervention recommendations in response to mission phase, subsystem safety classification, certification impact, and change control window status — reflects a fundamental architectural requirement that general-purpose refactoring automation frameworks are not designed to satisfy. ISR systems do not operate under static quality attribute priorities. During active collection missions, real-time throughput and determinism dominate all other considerations; a refactoring intervention that marginally improves long-term modifiability at the cost of transient scheduling disruption is categorically unacceptable regardless of its structural merit. During maintenance windows between mission cycles, that same intervention may be not only acceptable but urgently warranted.

The mission context vector encoding these operational state parameters transforms the refactoring engine from a structurally aware but operationally blind automation tool into a system that reasons about intervention timing as rigorously as it reasons about intervention substance. The experimental results confirm that this context-awareness translates into actionable differentiation: in the treated group analysis, the highest-impact remediation outcomes were associated with interventions applied during mission standby phases under open change control windows — precisely the conditions under which the engine assigns lowest RiskScore values and highest intervention priority. This alignment between model recommendations and observed remediation effectiveness validates the operational context encoding as a functionally meaningful architectural feature rather than a theoretical refinement without empirical consequence.

Implications for Practice

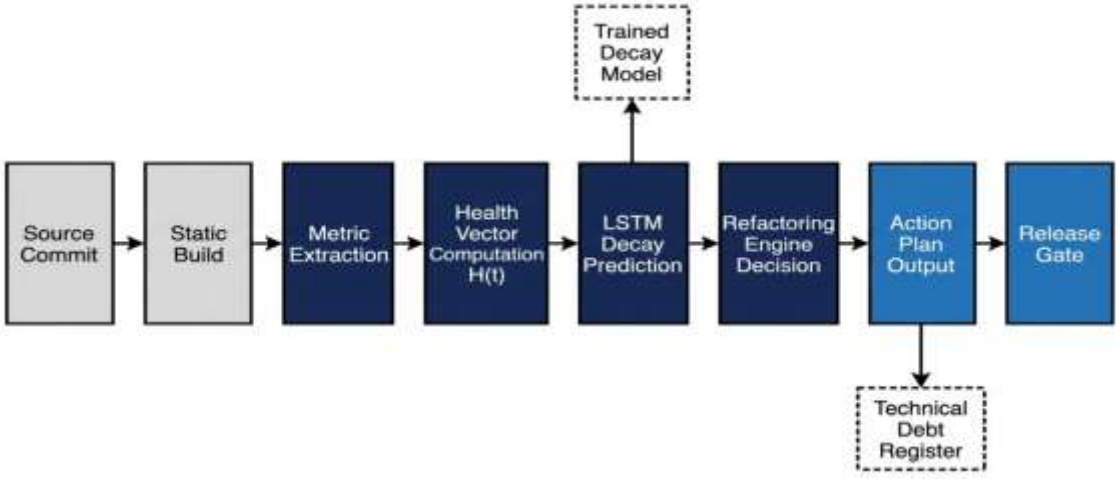

Figure: Integration architecture of the predictive refactoring framework within a DevSecOps continuous integration pipeline.

For software architects and engineers responsible for maintaining long-lived ISR platforms, the framework's most immediate practical implication is the viability of integrating continuous architectural health monitoring into existing configuration management and release engineering pipelines without requiring disruptive process re-architecture. The metric extraction toolchain described in Section 3.1 operates against compiled source baselines and version control histories using commercially available static analysis platforms, requiring no instrumentation of the runtime operational system. This non-invasive deployment profile is a prerequisite for adoption in defense software environments subject to rigorous change control and certification constraints.

Integration into DevSecOps pipelines — increasingly mandated across Department of Defense software programs under the DoD Enterprise DevSecOps Reference Design — is achievable by positioning the health vector computation and LSTM inference engine as automated quality gates within the continuous integration workflow. Each successful build triggers metric extraction; the predictive engine evaluates the resulting health vector against the trained decay model; and the refactoring engine generates a prioritized action plan that is surfaced to the software architect as a structured artifact within the program's issue tracking and technical debt register. This integration pattern requires no modification to the operational system and no disruption to existing release processes, representing an adoption pathway with manageable transition risk for programs operating under the constraints typical of fielded ISR platforms.

Limitations

Intellectual rigor demands explicit acknowledgment of the boundaries within which this study's findings are valid. Three limitations warrant particular attention.

First, the primary training corpus is dominated by AIMS-ISR architectural evolution data, introducing the risk of domain-specific overfitting. While the supplementary datasets from the Qualitas Corpus and the two additional defense-domain systems provide partial mitigation, the model's generalization performance on ISR platforms with substantially different architectural styles — highly distributed processing architectures, FPGA-accelerated sensor processing pipelines, or cloud-native ISR workloads emerging under the JADC2 framework — remains unvalidated. Claims of generalizability beyond AIMS-ISR class systems should be treated as provisional pending broader cross-platform validation.

Second, the computational overhead of continuous LSTM inference against a six-dimensional time-series health vector is non-trivial in resource-constrained deployment environments. While the analysis node configuration described in Section 4.1 is appropriate for program-office-level architectural monitoring, adaptation for deployment on embedded ground processing units operating under strict size, weight, and power constraints would require model compression, quantization, or distillation — techniques not evaluated in the current study.

Third, the scarcity of labeled *Critical*-class training examples — a structural consequence of the operational rarity of architecture-attributable anomalies in well-maintained systems — represents the most fundamental challenge to predictive model development in this domain. SMOTE-based oversampling provides a statistically defensible mitigation, but it cannot substitute for the representational richness of a genuinely larger *Critical*-class corpus. Programs willing to contribute sanitized architectural evolution data under information-sharing frameworks would directly address this limitation and strengthen the generalizability of successor models.

Future Work

Four concrete research extensions are prioritized. First, the integration of runtime telemetry — execution traces, memory allocation profiles, inter-process communication latency distributions — as additional feature dimensions beyond the static analysis metrics currently constituting the health vector would substantially enrich the model's observational basis and potentially reduce false negative rates on the *Critical* class toward the design target. Second, expansion of the evaluation corpus to encompass additional mission-critical domains — air traffic management systems, autonomous underwater vehicle control software, and space vehicle flight software — would test the framework's domain transferability and potentially reveal domain-specific metric weightings that improve cross-domain performance. Third, the current framework generates refactoring recommendations but delegates their execution to human engineers; the development of verified automated refactoring execution capabilities — applying formally verified behavior-preserving transformations under the engine's supervision — would close the loop from prediction to autonomous remediation, realizing the full potential of the predictive architecture assurance paradigm. Fourth, federated learning architectures that allow multiple defense programs to collaboratively train shared predictive models without exchanging raw source code or architectural artifacts would address the data scarcity limitation identified above while respecting the classification and proprietary constraints governing defense software assets — a research direction with significant practical and policy dimensions that extends well beyond the current study's scope.

CONCLUSION

This paper has presented a novel predictive software architecture evaluation model that integrates multi-dimensional architectural health monitoring with an adaptive, context-aware refactoring engine to detect and remediate structural decay in mission-critical ISR software systems before that decay precipitates operational failure. The framework was formalized, empirically validated against the AIMS-ISR architectural baseline, and benchmarked against three competing approaches spanning the spectrum from random intervention to reactive remediation and static threshold-based detection.

The principal finding is unambiguous: predictive, trajectory-aware architectural governance substantially outperforms all reactive and threshold-only alternatives across every operationally meaningful performance dimension. The LSTM-based decay model achieved a macro-averaged F1 score of 0.913 on the held-out

test set, delivered an average advance warning horizon of 5.3 release cycles prior to confirmed architecture-attributable anomalies, and enabled refactoring interventions that reduced architectural health metric deficiencies by an average of 42.7% — at a remediation cost ratio of 4:1 relative to post-failure corrective effort. These are not marginal improvements in engineering efficiency; they represent a structurally different relationship between software quality management and mission assurance, one in which failure is anticipated and forestalled rather than diagnosed and recovered.

The novelty of this contribution resides at the precise intersection of two previously disconnected bodies of work. Predictive decay modeling and automated refactoring have each advanced considerably as independent research domains; this paper demonstrates, for the first time in the context of operational defense software, that their integration within a unified, operationally context-sensitive framework produces capabilities that neither discipline achieves in isolation. The mission context engine — encoding safety classification, mission phase, certification impact, and change control constraints directly into refactoring priority logic — ensures that the framework's recommendations are not merely architecturally sound but operationally executable within the stringent governance environments that characterize fielded ISR platforms.

The broader implication of this work extends beyond the ISR domain to the full class of long-lifecycle, mission-critical software systems on which modern defense and aerospace capability depends. As these systems grow in complexity, accumulate operational history, and absorb successive generations of capability insertion, the structural forces driving architectural decay will only intensify. The reactive paradigm — waiting for failure before investing in structural remediation — is no longer a defensible engineering posture in systems where failure carries strategic consequences. Proactive architectural stewardship, grounded in rigorous predictive modeling and operationally calibrated automated intervention, is not a research aspiration; it is an engineering imperative. This paper provides a validated, transferable framework for making that imperative operational — and in doing so, advances the foundational engineering capability on which the reliability and mission success of future defense and aerospace systems will increasingly depend.

REFERENCES

Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.

Avgeriou, P., Kruchten, P., Nord, R. L., Ozkaya, I., & Seaman, C. (2016). Managing technical debt in software engineering. *Dagstuhl Reports, 6*(4), 110–138. https://doi.org/10.4230/DagRep.6.4.110

Clements, P., Kazman, R., & Klein, M. (2002). *Evaluating software architectures: Methods and case studies*. Addison-Wesley Professional.

Cunningham, W. (1992). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger, 4*(2), 29–30. https://doi.org/10.1145/157709.157715

Feiler, P. H., Lewis, B. A., & Vestal, S. (2006). The SAE architecture analysis and design language standard: A basis for model-based architecture-driven embedded systems engineering. *INCOSE International Symposium, 16*(1), 1–16. https://doi.org/10.1002/j.2334-5837.2006.tb02711.x

Fowler, M. (2018). *Refactoring: Improving the design of existing code* (2nd ed.). Addison-Wesley Professional.

Gorton, I. (2011). *Essential software architecture* (2nd ed.). Springer. https://doi.org/10.1007/978-3-642-19176-3

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software, 29*(6), 18–21. https://doi.org/10.1109/MS.2012.167

Lehman, M. M., & Belady, L. A. (1985). *Program evolution: Processes of software change*. Academic Press.

Luckham, D. C., & Vera, J. (1995). An event-based architecture definition language. *IEEE Transactions on Software Engineering, 21*(9), 717–734. https://doi.org/10.1109/32.464548

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering, 30*(2), 126–139. https://doi.org/10.1109/TSE.2004.1265817

Mkaouer, M. W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., & Ouni, A. (2016). Many-objective software remodularization using NSGA-III. *ACM Transactions on Software Engineering and Methodology, 24*(3), 1–45. https://doi.org/10.1145/2729974

Murphy-Hill, E., Parnin, C., & Black, A. P. (2012). How we refactor, and how we know it. *IEEE Transactions on Software Engineering, 38*(1), 5–18. https://doi.org/10.1109/TSE.2011.41

Nugroho, A., Visser, J., & Kuipers, T. (2011). An empirical model of technical debt and interest. *Proceedings of the 2nd Workshop on Managing Technical Debt*, 1–8. https://doi.org/10.1145/1985362.1985364

Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A. (2018). On the diffuseness and the impact on maintainability of code smells: A large scale empirical investigation. *Empirical Software Engineering, 23*(3), 1188–1221. https://doi.org/10.1007/s10664-017-9535-z

Seaman, C., & Guo, Y. (2011). Measuring and monitoring technical debt. *Advances in Computers, 82*, 25–46. https://doi.org/10.1016/B978-0-12-385512-1.00002-5

Tornhill, A. (2018). *Software design X-rays: Fix technical debt with behavioral code analysis*. Pragmatic Bookshelf.

Yamashita, A., & Moonen, L. (2013). Exploring the impact of inter-smell relations on software maintainability: An empirical study. *Proceedings of the 35th International Conference on Software Engineering*, 682–691. https://doi.org/10.1109/ICSE.2013.6606614

Zdun, U., Navarro, E., & Leymann, F. (2007). Ensuring and assessing architecture conformance to microservice decomposition patterns. *IEEE Transactions on Software Engineering, 45*(8), 733–749. https://doi.org/10.1109/TSE.2007.70771