International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P117
Eureka Vision Publication | Volume 4, Issue 3, 164-170, 2023

Original Article

Integrating Site Reliability Engineering SRE Principles into Enterprise Architecture for Predictive Resilience

Siva Kantha Rao Vanama
Cloud Architect Deloitte Tampa, Florida, USA.

Abstract - *Modern enterprises increasingly depend on complex distributed software systems where small faults can cascade into large customer impact. Site Reliability Engineering provides a disciplined approach to reliability through explicit service level objectives, error budgets, automation, incident response and continuous learning. Enterprise Architecture provides an enterprise-wide design view that connects business capabilities, information flows and technology platforms. This paper proposes an integrated framework that makes reliability a first-class architecture concern and that links architecture decisions to runtime evidence. The framework introduces an artifact mapping between Enterprise Architecture models and SRE primitives such as service level indicators, service level objectives and error budgets. It also defines a predictive resilience loop that combines observability telemetry with architecture context and change signals to anticipate degradation risk before user impact occurs. The paper synthesizes related work on resilience, observability, trace and log anomaly detection and interpretable root cause analysis then proposes implementation patterns for SLO hierarchies, drift detection, policy driven release governance and chaos experiments. Finally, it defines evaluation metrics and an illustrative enterprise scenario that demonstrates how predictive signals can trigger targeted governance actions and architecture updates.*

Keywords - *Site Reliability Engineering, Enterprise Architecture, Observability, Predictive Resilience, Aiops, Service Level Objectives.*

1. Introduction

Enterprises are under sustained pressure to ship software faster while maintaining high availability and predictable performance. As systems scale, variability rather than average behavior becomes the dominant driver of user experience. Tail latency effects mean a small fraction of slow requests can define perceived performance and can create cascading timeouts across dependencies [1]. The shift toward microservices, continuous delivery and shared platforms has increased the number of service interactions, configuration surfaces and change events. In this environment, production incidents are often caused by interaction effects between components rather than a single failing component.

Site Reliability Engineering, or SRE, emerged to manage this complexity by treating operations as a software engineering problem. SRE emphasizes explicit reliability targets, strong measurement, automation, incident response discipline and continuous learning. Error budgets and service level objectives provide a shared mechanism that balances reliability and feature velocity by making the cost of unreliability visible and actionable [3]. The evolving SRE engagement model also describes how reliability teams partner with product teams using explicit criteria, shared accountability and measurable outcomes [2].

In parallel, Enterprise Architecture, or EA, provides enterprise-wide methods to align business strategy, information systems and technology platforms. EA governance research stresses decision rights, roles and processes that assure consistency and timeliness of architecture outputs [4]. Yet many EA programs historically focused on static target state designs and periodic reviews while digital delivery requires decisions to stay connected to runtime outcomes. When EA and SRE remain disconnected, architecture decisions may be made without operational evidence and reliability work may optimize local service issues without addressing systemic architecture drivers.

This paper addresses the gap by integrating SRE principles into EA, so reliability becomes an explicit architecture attribute with traceable contracts, evidence and controls. We also extend the integration with predictive resilience, meaning the capability to anticipate degradation risk and steer decisions before incidents affect users. Predictive resilience draws from observability and AIOps research that uses logs, traces and metrics for anomaly detection, correlation and diagnosis [7]. It also aligns with resilience research that views resilience as an adaptive capability supported by agility, connectivity and learning [9], [11].

The contributions of this paper are fourfold. First, we define a mapping between EA artifacts and SRE primitives including service level indicators, service level objectives and error budgets. Second, we propose a governance loop that continuously validates architecture assumptions using runtime telemetry and predictive analytics. Third, we present

implementation patterns that combine SLO decomposition, trace and log anomaly detection and interpretable root cause analysis. Fourth, we define evaluation metrics that cover reliability outcomes, architecture governance effectiveness and predictive model quality.

2. Background and Related Work

2.1. Site Reliability engineering concepts

SRE formalizes reliability as a product feature with measurable objectives. A service level indicator is a quantitative measure of service behavior such as availability, latency, correctness or durability. A service level objective sets a target for the indicator over a defined time window. The error budget is the allowed unreliability implied by the objective. It becomes an operational control that balances innovation and reliability because it provides a shared view of how much risk has been consumed [3]. When error budget burn accelerates, release controls tighten and engineering effort shifts toward reliability work until budgets return to policy levels.

SRE also emphasizes production readiness and toil reduction. Toil is manual operational work that is repetitive, lacks enduring value and can be automated. By investing in automation and safer change mechanisms, SRE reduces the cognitive load of on call response and creates predictable incident handling. The SRE engagement model highlights partnership patterns where SRE teams help define reliability targets, build shared tooling and evolve responsibilities with product teams as services mature [2].

2.2. Enterprise Architecture governance and resilience

EA connects business goals with technology and provides methods to design, standardize and govern enterprise systems. Governance is critical because architecture is not only a design artifact but also a decision process. Research on EA governance emphasizes clear decision rights and processes that assure consistency and timeliness of EA outputs [4]. Modern enterprises need governance that is risk aware and evidence driven rather than purely document driven.

Resilience in enterprises has been studied across organizational and engineering perspectives. Extended enterprise resilience has been framed around attributes such as agility, flexibility, adaptability and connectivity that support response and recovery in complex environments [9]. Engineering oriented resilience principles emphasize avoidance, survival and recovery and argue that systems should withstand disruptions with minimal degradation and restore function rapidly [11]. Within EA, resilience is increasingly treated as a critical quality attribute. Systematic reviews identify that EA resilience research is still developing and that practical methods are needed to incorporate resilience characteristics into EA models and decisions [5]. Methods such as resilience by design propose guidance for representing resilience characteristics in EA models and for linking them to design choices [6].

2.3. Observability and AIOps foundations for predictive resilience

Predictive resilience depends on observability, meaning the ability to infer internal system states from external signals. Distributed tracing enables request level visibility across microservices and supports dependency mapping and anomaly detection. Research on trace based anomaly detection demonstrates methods for large scale trace analysis that detect anomalies and localize likely faulty services in microservice systems [13]. Group wise trace methods reduce heterogeneity by grouping traces into comparable sets and can improve detection quality [14]. Combined trace and log approaches such as DeepTraLog show that integrating multiple telemetry sources can improve anomaly detection in complex systems [15]. Log based anomaly detection also remains important. DeepLog models log event sequences and detects anomalies by learning normal patterns and flagging deviations [16].

AIOps surveys describe how machine learning can support event correlation, anomaly detection and root cause analysis. They also highlight emerging interest in large language models for operations assistance while stressing the need for governance and evaluation [7]. Interpretable root cause analysis methods that learn weighted causal graphs can improve trust because they provide human readable explanations that align with incident reasoning [8]. Graph neural network based approaches for root cause analysis further model dependency structure and anomaly propagation in microservice systems [17]. These streams suggest that prediction and early warning are feasible but enterprise adoption requires strong context, clear decision rights and integration with architecture processes.

3. Problem Statement and Design Goals

Enterprises often operate with two partially disconnected views of the same system. The EA view describes designed architecture, intended business capability support and target state plans. The operations view describes runtime behavior, incidents, service ownership and on call response. When these views are disconnected, three problems commonly appear.

First, reliability targets are inconsistent. Teams choose availability or latency targets without an enterprise level taxonomy that relates targets to business criticality. This creates mismatched risk tolerance across dependencies and encourages local optimization. Second, architecture decisions are rarely validated by operational evidence. A design review may approve

patterns such as service mesh routing, asynchronous messaging or caching without measuring whether the pattern actually reduces incidents, improves recovery or controls tail latency. Third, incident learning does not systematically update architecture. Post incident actions are captured in tickets or runbooks but architecture artifacts such as dependency models, standards and reference architectures remain stale.

Predictive resilience adds a fourth challenge. Many enterprises deploy anomaly detection tools but output is noisy and not connected to decision rights. Alert volume increases without improving reliability outcomes. Prediction becomes useful only when it is tied to specific actions such as change controls, capacity adjustments or targeted architecture review.

The design goals for an integrated EA and SRE approach are therefore: 1. Traceability. Reliability objectives should be traceable from business capabilities to services to operational signals and controls. 2. Evidence driven governance. Architecture governance should incorporate runtime evidence and should update artifacts based on operational outcomes. 3. Actionable prediction. Predictive signals should be connected to defined actions that reduce risk before incidents affect users. 4. Scalability. The approach should scale across many teams with clear ownership, minimal bureaucracy and automation where possible.

4. Enterprise Architecture and SRE Predictive Resilience Framework

4.1. Framework overview

We propose the Enterprise Architecture and SRE Predictive Resilience Framework abbreviated as EASPRF. The framework treats SRE constructs as first class EA artifacts and introduces a closed loop that couples architecture governance with reliability outcomes and predictive analytics. EASPRF is organized into five layers.

Layer 1 is Business Capability and Value. This layer defines capabilities, customer journeys and business objectives including impact models and acceptable risk levels. Layer 2 is Architecture Intent. This includes service boundaries, data domains, integration patterns, platform choices and quality attribute assumptions. Layer 3 is Reliability Contract. This defines service level indicators, service level objectives and error budget policies for services and journeys. Layer 4 is Observability and Evidence. This layer standardizes telemetry instrumentation, metadata and correlation so runtime evidence can be connected to architecture entities. Layer 5 is Prediction and Control. This layer performs risk estimation and triggers defined controls in delivery and governance processes.

EASPRF is built on the principle that architecture decisions must be continuously validated against operational outcomes. It replaces the one way flow from architecture to implementation with a closed loop where telemetry feeds prediction and where prediction triggers actions that update architecture artifacts and delivery controls.

4.2. Mapping EA artifacts to SRE primitives

EASPRF provides a mapping so reliability becomes a traceable architecture property. Business capabilities map to end to end user journeys. For each journey, an experience SLO is defined, for example checkout success rate and checkout latency percentile. The experience SLO is decomposed into service SLOs using dependency graphs and error propagation assumptions. Work on SLA decomposition provides methods to translate higher level objectives into system level thresholds [18]. We adapt this by treating the decomposition itself as an EA governed artifact that is updated when architecture changes.

EA application components map to runtime services and workloads. Each service has an SLI set such as availability, latency, correctness and throughput and an SLO target for each SLI. SLOs connect to error budgets that drive release controls, prioritization and reliability investment decisions [3]. EA integration views map to runtime dependencies. Dependencies are learned from traces, service discovery and network telemetry. Drift between designed dependencies and observed dependencies is tracked as an architecture risk indicator that can signal hidden coupling or unapproved integration paths.

EA standards and reference architectures map to reliability patterns such as timeouts, retries, circuit breakers, bulkheads, rate limiting and progressive delivery. These patterns are validated using production evidence and controlled experiments. Chaos engineering proposes fault injection to validate resilience assumptions and improve readiness [12]. Within EASPRF, controlled experiments become architecture validation evidence and are used to refine both patterns and predictive models.

4.3. Predictive resilience loop

The predictive loop connects operations evidence back to architecture and forward to decision controls. It operates continuously for telemetry collection and on a cadence for governance actions.

Step 1 observe. Metrics, logs and traces are collected with consistent tags for service name, owner, environment and deployment identifier. Step 2 model. A dependency graph is derived from traces and service discovery. Anomaly detectors operate on service time series, trace structure and log sequences. Large scale trace analysis methods support detecting anomalies and localizing them to candidate services [13]. Step 3 predict. A risk model estimates probability of SLO violation

or incident within a near term horizon such as the next hour or day. Inputs include anomaly scores, dependency criticality, recent change features and error budget burn rate. Step 4 decide. If risk crosses thresholds, defined controls are triggered. Controls include progressive delivery escalation, automated rollback, capacity adjustment or targeted architecture review. Step 5 learn. After incidents or near misses, post incident analysis updates architecture artifacts such as dependency models, standards and SLO decompositions and retrains predictive models with labeled outcomes.

This loop operationalizes resilience attributes such as agility and connectivity by making adaptation explicit and measurable [9]. It also aligns with resilience principles that emphasize survival and rapid recovery under disruption [11]. The novelty is that adaptation is governed and connected to EA artifacts rather than being only an operational practice.

5. Reference Implementation Patterns

5.1. Reliability contract design and SLO hierarchies

Enterprises need a consistent method to define SLOs at multiple levels. We recommend a three level hierarchy. Level 1 journey SLOs capture end user outcomes. Level 2 domain service SLOs capture reliability of major capabilities. Level 3 platform SLOs capture reliability of shared infrastructure such as identity, messaging and data platforms. Each level has clear ownership and a policy for how error budgets influence change.

To implement decomposition, a graph based approach is used. A dependency graph is built from traces. SLO budgets are allocated to components based on their contribution to failure probability and latency. This approach is inspired by work on translating service objectives to system thresholds [18]. The decomposition is reviewed by the architecture council and updated when dependencies change. This prevents the common problem where a journey SLO is declared but there is no mechanism to ensure component budgets align with it.

5.2. Observability as an EA standard

Observability must be treated as an architecture standard rather than an optional tool choice. We propose an EA owned observability reference architecture with four concerns.

First, instrumentation standards define which spans, logs and metrics each service must emit and what attributes must be present. Second, data pipeline patterns define ingestion, storage and query with explicit reliability requirements for the observability system itself. Third, metadata standards define how ownership, environment, region and change identifiers are attached so telemetry can be linked to EA entities and to deployment events. Fourth, analytics standards define how SLO computation, anomaly detection and incident workflows consume telemetry.

Research on trace based anomaly detection shows the importance of consistent trace structure and efficient feature extraction [13], [14]. Combined trace and log approaches further require consistent correlation between traces and logs [15]. These findings motivate enterprise wide telemetry schemas and strict correlation identifiers.

5.3. Predictive detection and diagnosis pipeline

A predictive pipeline is organized into three analytic stages.

- Stage 1 anomaly detection identifies unusual behavior. For logs, sequence models learn normal event patterns and flag deviations as anomalies as shown by DeepLog [16]. For traces, graph and sequence representations detect anomalies in call structure and latency distributions. Large scale trace analysis methods show how to localize anomalies to services in a microservice graph [13]. Group wise approaches improve quality by comparing traces within similar groups [14]. DeepTraLog demonstrates that combining traces and logs can improve detection and localization by capturing complementary signals [15].
- Stage 2 correlation and impact estimation groups anomalies by time, service and dependency then estimates business impact using capability mappings. Tail latency effects mean variance changes can cause user impact even when averages remain stable [1]. Impact estimation therefore focuses on percentiles and error rates. It also estimates blast radius using dependency criticality and traffic volume.
- Stage 3 root cause hypothesis generation produces interpretable hypotheses. Causal graph methods that learn weighted event graphs can support interpretable root cause analysis for microservices [8]. Graph neural network based methods can also support diagnosis by modeling dependency structure and anomaly propagation [17]. Hypotheses are linked to architecture artifacts such as recent changes, dependency drift, SLO budget mismatches or policy violations.

5.4. Reliability centered change governance using error budgets

Error budgets provide a practical control mechanism. When burn is within policy, releases proceed with standard safeguards. When burn accelerates, progressive delivery is required and higher risk changes are slowed. When burn exceeds a threshold, releases pause and teams focus on reliability work. This approach aligns with SRE practice and provides a measurable basis for governance decisions [3]. Release engineering research provides additional patterns for disruption free change delivery. Zero downtime release techniques show how to reduce disruption during large scale rollouts through careful

load balancing and state handling [20]. While not all enterprises operate at hyperscale, the underlying principle of isolating users from disruptive events can be encoded as an architecture standard for session management, database migrations and traffic shifting.

5.5. Resilience testing and chaos experiments

Predictive models benefit from diverse failure observations. Chaos engineering introduces controlled fault injection to validate assumptions, train teams and improve system resilience [12]. Within EASPRF, chaos experiments are planned based on predicted risk hotspots and dependency criticality. Results are recorded as evidence linked to EA patterns and updated runbooks. This supports engineered system resilience principles that emphasize survival and rapid recovery [11].

6. Evaluation Metrics and Illustrative Scenario

6.1. Reliability outcome metrics

Reliability outcomes are evaluated using user journey availability, journey latency percentiles and correctness rates. Operational metrics include mean time to detect, mean time to restore, incident frequency and change failure rate. SLO compliance is tracked via error budget burn rates which provide an interpretable summary of reliability consumption [3].

6.2. Architecture and governance metrics

Architecture governance effectiveness is measured by dependency drift and artifact freshness. Dependency drift measures the fraction of observed dependencies not present in the designed model and the fraction of designed dependencies not observed. Artifact freshness measures time from an operational learning event to an EA artifact update. Governance efficiency is measured by decision cycle time and the proportion of decisions triggered automatically by policy rather than by manual review.

EA governance research emphasizes timeliness and consistency of architecture outputs [4]. EASPRF operationalizes this through measurable update loops, standards adoption metrics and reduction of repeated incident classes linked to architecture patterns.

6.3. Predictive model metrics

Predictive quality is measured using precision, recall and area under the ROC curve for incident risk prediction. For probabilistic output, calibration and Brier score are used. Model drift is measured by changes in performance over time as services evolve. Predictive resilience modeling research suggests that resilience metrics and recovery times can be predicted with regression methods and evaluated using error measures [21]. EASPRF applies this by predicting SLO violation risk and expected recovery time under degraded conditions.

6.4. Illustrative scenario

Consider an enterprise retail platform where checkout depends on identity, pricing, inventory and payment services. EA defines the checkout capability and sets a journey SLO of 99.9 percent success and a latency target at the ninety fifth percentile. The SLO is decomposed into service SLOs with explicit error budgets. Observability instruments spans across services and tags traces with deployment identifiers.

A pricing release introduces an increase in tail latency due to a new synchronous dependency from pricing to inventory. Trace anomaly detection flags increased variance in pricing spans and dependency drift detection identifies the new call that was not present in the designed model. The risk model predicts a high probability of checkout latency SLO violation within the next hour based on the anomaly trend, error budget burn acceleration and traffic ramp. Governance rules trigger an automatic canary rollback for the pricing release and open a targeted architecture review.

During review, teams discover that the synchronous call was added to meet a feature deadline. The architecture council updates the integration standard for pricing to require asynchronous enrichment patterns and tighter timeout policies for critical paths. The SLO decomposition is updated to reflect the dependency risk and the predictive model is retrained with the labeled near miss outcome. The outcome is no customer visible incident and a concrete architecture improvement linked to measurable telemetry evidence.

7. Adoption Guidance

7.1. Roles and operating model

Adoption requires clear ownership and decision rights. We recommend a joint Architecture and Reliability Council with enterprise architects, SRE leads and platform engineering. The council owns reliability taxonomy, observability standards and governance policies. Service owners remain accountable for their SLOs and reliability work. Architects remain accountable for architecture decisions and for maintaining designed models and standards.

7.2. Data and tooling considerations

Telemetry pipelines must be reliable and scalable. Surveys of real time anomaly detection highlight the importance of handling volume, velocity and data quality in monitoring pipelines [19]. This implies careful design for sampling, retention and query efficiency as well as resilience of the observability stack itself. Metadata management is also essential so telemetry can be joined with architecture context, change data and ownership information.

7.3. Common pitfalls

A frequent pitfall is treating SLOs as compliance targets rather than negotiated contracts linked to user value. Another pitfall is attempting predictive operations without a clean service taxonomy and ownership model so signals cannot route to action. A third pitfall is governance overload where reviews slow delivery. EASPRF mitigates this by using automated controls driven by error budgets and predicted risk so manual governance effort scales with risk rather than with project count.

8. Discussion and Future Work

The framework integrates architecture and operations but limitations remain. Prediction quality can be limited by sparse labels, changing architectures and non stationarity of workloads. Interpretability remains important for trust and adoption which motivates causal and graph based methods [8], [17]. Security is also a critical dimension of resilience. EA modeling for cybersecurity analysis proposes structured approaches to analyze security risk using EA models [23]. Combining those methods with SRE telemetry can create a unified risk view where security anomalies and reliability anomalies are interpreted together.

Future work includes empirical validation across industries, refined SLO decomposition algorithms and improved blast radius estimation. AIOps surveys note opportunities to use large language models for operations assistance but emphasize the need for governance, evaluation and safe integration [7]. Controlled use of language models for summarization, runbook assistance and architecture knowledge retrieval could be explored within EASPRF with strict guardrails and measurable outcomes.

9. Conclusion

Enterprises need both architectural intent and operational discipline to achieve reliable software delivery at scale. SRE provides mechanisms to define reliability contracts, balance innovation and reliability and institutionalize learning. EA provides the method and governance to align business capabilities, systems and platforms. By integrating these approaches and adding predictive resilience through observability and analytics, enterprises can connect design decisions to runtime evidence and act before user impact occurs. The proposed framework makes reliability traceable across architecture artifacts, introduces a closed loop that continuously validates assumptions and connects prediction to actionable controls. This combination supports enterprise resilience by design and by operation, enabling organizations to deliver faster while sustaining dependable customer experience.

References

[1] J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74 to 80, 2013, doi: 10.1145/2408776.2408794.

[2] B. Treynor et al., "The Evolving SRE Engagement Model," Communications of the ACM, 2017, doi: 10.1145/3080202.

[3] S. Sha et al., "Error Budgets and SLOs," Communications of the ACM, 2019, doi: 10.1145/3369756.

[4] R. Winter and R. Fischer, "Enterprise Architecture Governance," in Proceedings of the 2008 ACM Symposium on Applied Computing, 2008, doi: 10.1145/1363686.1363820.

[5] Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016). Relationship between DevOps and lean: A multi-case study of software organizations. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE), 1–10. https://doi.org/10.1145/2915970.2915996

[6] A. Aldea et al., "Enterprise Architecture Resilience by Design: A Method and Guidance," in 2021 IEEE EDOC Workshops, 2021, doi: 10.1109/EDOCW52865.2021.00044.

[7] Lyu, Y., Rajbahadur, G. K., Lin, D., Chen, B., Zhen, M., & Jiang. (2022). Towards a consistent interpretation of AIOps models. arXiv. https://doi.org/10.48550/arXiv.2202.02298

[8] Xin, R., Chen, P., & Zhao, Z. (2022). CausalRCA: Causal inference based precise fine-grained root cause localization for microservice applications. arXiv. https://arxiv.org/abs/2209.02500

[9] O. Erol, B. J. Sauser and M. Mansouri, "A Framework for Investigation into Extended Enterprise Resilience," Enterprise Information Systems, vol. 4, no. 2, pp. 111 to 136, 2010, doi: 10.1080/17517570903474304.

[10] T. J. Vogus and K. M. Sutcliffe, "Organizational Resilience: Towards a Theory and Research Agenda," in 2007 IEEE International Conference on Systems, Man and Cybernetics, 2007, doi: 10.1109/ICSMC.2007.4414160.

[11] Gunda, S. K. G. (2023). The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126-129. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113

[12] A. Basiri et al., "Chaos Engineering," IEEE Software, vol. 33, no. 3, pp. 35 to 41, 2016, doi: 10.1109/MS.2016.60.

[13] Y. Wu et al., "Large Scale Trace Analysis for Microservice Anomaly Detection and Localization," in Proceedings of the ACM SIGKDD Conference, 2022, doi: 10.1145/3531056.3542765.
[14] Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2010). Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09) (pp. 117–132). https://doi.org/10.1145/1629575.1629588
[15] C. Zhang et al., "DeepTraLog: Trace Log Combined Microservice Anomaly Detection," in Proceedings of the ACM International Conference on Software Engineering, 2022, doi: 10.1145/3510003.3510180.
[16] M. Du et al., "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2017, doi: 10.1145/3133956.3134015.
[17] Z. Li et al., "Root Cause Analysis of Anomalies Based on Graph Neural Network in Microservice Systems," International Journal of Software Engineering and Knowledge Engineering, 2022, doi: 10.1142/S0218194022500395.
[18] Y. Chen et al., "SLA Decomposition: Translating Service Level Objectives to System Level Thresholds," in 2007 International Conference on Autonomic Computing, 2007, doi: 10.1109/ICAC.2007.36.
[19] R. A. Ariyaluran Habeeb et al., "Real Time Big Data Processing for Anomaly Detection: A Survey," International Journal of Information Management, vol. 45, pp. 289 to 307, 2019, doi: 10.1016/j.ijinfomgt.2018.08.006.
[20] U. Naseer et al., "Zero Downtime Release: Disruption Free Load Balancing of a Multi Billion User Website," in Proceedings of ACM SIGCOMM, 2020, doi: 10.1145/3387514.3405885.
[21] Ye, X., & Reza, M. (2020). Statistical learning and regression techniques for resilience assessment in infrastructure systems. Reliability Engineering & System Safety, 198, 106848. https://doi.org/10.1016/j.ress.2020.106848
[22] S. Fedushko et al., "Cloud Platforms and Enterprise Data: Components, Principles and Approaches," Applied Sciences, 2020, doi: 10.3390/app10249112.
[23] Eljabiri, A. R., Selim, G., & Schmidt, R. (2018). Leveraging enterprise architecture for cybersecurity management: A framework and its application. Journal of Information Security and Applications, 39, 41–52.
[24] Hillmann et al., "Integrated Enterprise Resilience Architecture Framework for Surviving Strategic Disruptions," Enterprise Risk Management, 2018, doi: 10.5296/erm.v4i1.13715.