Architectural Evolution in Practice: Comparing Monolithic and Microservices Migration Approaches

Tarun Kalwani *tarun.kalwani17@gmail.com*
Independent Researcher
Atlanta, Georgia, United States

Abstract

The world-wide migration to microservices from monolithic is driven by demands for scalability, agility, and maintainability.1 Migration as such is however afflicted with technical and organizational issues.2 The choice of the optimal migration approach will be the recipe for success, but little empirical data to compare it with exists. The research proposes a comparative investigation of the three most debated migration approaches: Big Bang, Strangler Fig, and Parallel Run. We are comparing their impact on some of the most critical performance metrics like downtime, migration expense, project time, and system resilience. The research is done using simulated data of 471 small to large enterprise migration projects. Python Pandas libraries were utilized in data handling and Scikit-learn for use of the regression model for strategy selection and project performance forecasting. Deployment trends were emulated by Docker and Kubernetes for ensuring resiliency capability. Following our findings, in some cases Big Bang style is quicker, the Strangler Fig approach is always lower in operational risk and higher in resilience over the long term but at the cost of longer project duration. Parallel Run strategy is the most secure but at astronomically high cost of infrastructure. This paper suggests a quantitative method in order to enable organizations to take the correct migration strategy depending on risk tolerance and business requirements.

This study addresses the research question: "How do Big Bang, Strangler Fig, and Parallel Run migration strategies differ in terms of cost, risk, resilience, and implementation effort when applied to typical monolithic-to-microservices transformations?" The findings provide comparative, data-driven insights that can help architects and engineering leaders make informed decisions based on risk tolerance, operational continuity needs, and economic constraints.

Keywords: Microservices, Monolithic Architecture, Migration Styles, Strangler Fig Pattern, Refactoring Software.

1. INTRODUCTION

Monolithic pattern has been the norm for software system architecture with single codebases adopting the user interface, business logic, and database management layers according to Chen et al. (2020). The design philosophy in applying such an implementation was rather efficient at the early days of software development and when software complexity was low and fewer resources were restricting the demand for distributed systems according to Camilli et al. (2022). Its ease of deployment and simplicity in implementation made it extremely appropriate for small applications and start-ups with minimal overhead in maintenance and long development cycles, according to Jamshidi et al. (2018). But with more and more software applications arising because of mounting needs of scalability and adaptability, monolithic systems brittleness as designs were functioning in the role of an innovation barrier, as per Francesco et al. (2019).One of the severe limitations of monolithic systems is that they are composed of tightly-coupled modules where even a tiny change is dangerous and expensive, as quoted by Ayas et al. (2021). It is expensive to modify one component, with re-deployment of the entire system, for the expense of increased downtime and inflexible dependencies, as argued by Hassan et. al. (2020). Further, the same level of stiffness enables technology lock-in, where it is difficult to migrate to

newer frameworks or tools with extensive reengineering, as argued by Tahir et al. (2020). Also, scaling issues accompany another important bottleneck—monolithic apps scale in one mass, and consequently, organizations over-provision resources, the cost of running wild, as implied by Soldani et al. (2018). The said above constraints came into being with microservices architecture, where systems are divided into loosely connected, independently deployable services that each perform one isolated business task, as implied by Ntentos et al. (2021).

Its advantages—flexibility, scalability, and fault isolation—obstacles aside, microservices migration from monolithic systems is risky and complex, Lenarduzzi et al. (2020) concludes. Migration effort translates to business continuity disruption, cost escalation, and operating complexity escalation, Balalaie et al. (2018) finds. This article thus presents comparative analysis grounded on evidence of two most widely used migration methods: the "Big Bang" rewrite and the "Strangler Fig" step-by-step migration approach promoted by Auer et al. (2021). Measurement in terms of cost-effectiveness, downtime, and dependency risk as metrics, the study is seeking to provide practical guidelines to companies ready to implement an optimum migration strategy, as axiomatized by Francesco et al. (2019).

2. LITERATURE REVIEW

Software architecture evolution from monolithic to modular and service-oriented architecture styles is a retropation of the industry's evolution towards deployability, scalability, and reliability, as theorized by Chen et al. (2020). Early in its history, the monolithic paradigm came with inherent simplicity that enabled programmers to make use of packaged systems that supported global memory space sharing and common execution contexts, as discussed by Jamshidi et al. (2018). The monolithic styles grew cumbersome when large amounts of code had accumulated that offered maintenance chokepoints, brittle testability, and slow release cycles, as posed by Hassan et al. (2020). Other than that, CI/CD pipeline development matured an extra step further to even improve the shortcomings of monolithic systems that cannot mature to agile iteration or sandbox test environments, as explained in Ayas et al. (2021).To address these drawbacks, microservices architecture has been an answer to the module and scalability problem as proposed in Camilli et al. (2022). The software components of the architecture are unbundled into separate services and each of them addresses a specific business capability with its own development, deployment, and scale as described in Lenarduzzi et al. (2020). The unbundling provides greater flexibility in the sense that teams can utilize other programming languages, frameworks, and databases suitable for each application of the service, as explained in Soldani et al. (2018). Other than that, microservices are inherently abhorrent to DevOps and cloud-native cultures that prioritize continuous delivery and further support fault tolerance, as presented by Balalaie et al. (2018).

Alternatively, a monolithic system to microservices systems migration is of utmost engineering and managerial importance, as presented by Tahir et al. (2020). Specific master migration techniques have been suggested based on studies. The "Big Bang" migration is complete re-definition of the system, new beginning but with extremely high risk of failure because it is in its nature costly and time-consuming to develop, proposed by Auer et al. (2021). The "Strangler Fig" method, on the other hand, accommodates incremental change where the new functionality arrives as microservices but the old monolith stays in place until it gets replaced, proposed by Francesco et al. (2019). This incremental approach minimizes downtime and risk at the cost of communication interface coordination and data consistency with very conservative care, as put forward in Ntentos et al. (2021).

"Branch by Abstraction" is another approach that introduces abstraction layers into the monolith in a way that modules are unbundled prior to being shipped to microservices in order to facilitate enhanced evolution without compromising the current functionalities, as described by Jamshidi et al. (2018). But among the most significant long-term issues in all these approaches is database decomposition since monolithic structures usually rely on centralized databases that won't break into fragments without sacrificing data integrity, as Lenarduzzi et al. (2020) argued. Though other

research, as referenced by Hassan et al. (2020) and Ayas et al. (2021), have tried these methods separately from each other, there are none that use comparative, empirical research with controlled levels of cost, controlled levels of downtime, and system complexity, as suggested by Soldani et al. (2018). This research provides this gap by providing quantitative comparative analysis of migration methods that will compel organizations to data-driven architecture change decisions, as suggested by Tahir et al. (2020).

Critical Assessment of Prior Work and Identified Gaps
Although there is extensive existing literature describing detailed migration patterns, such as Big Bang, Strangler Fig, and Parallel Run, much of this literature remains conceptual or at best based on isolated case studies rather than on structured comparative evidence. Prior works, such as Soldani et al. (2018) and Di Francesco et al. (2019), which outline the challenges and benefits of microservices adoption, do not empirically quantify trade-offs across different strategies. Similarly, other studies focused on technical debt, team coordination, or code smells give important context but are related only indirectly to evaluating different migration strategies. This scarcity of aggregated, cross-strategy comparison is a clearly perceived gap that the current study aims to fill. However, in contrast to these prior works, which rely on either real-world data or qualitative interviews, the research here utilizes simulated project-level data to approximate patterns observed in the industry. We can develop an improved justification to better detail why synthetic data was used as a practical surrogate due to a lack of large-scale public datasets and demonstrate where the study contributes relative to existing empirical attempts.

2.1 Methodology

Research Design, Data Collection, and Analytical Approach
This research design is quantitative and deductive. The deductive orientation involves testing theoretically established assumptions about the monolith-to-microservices migration strategy regarding expected differences in cost, downtime, failure rates, and resilience against a controlled dataset. Because no large-scale datasets have been publicly shared in the given domain, the data collection consists of generating a synthetic dataset of 471 projects. The synthetic data was generated to approximate real-world distributions of variables such as monolith complexity (LOC), technical debt, team size, and deployment frequency based on ranges and correlations from prior industry studies and published migration post-mortems. In this way, this study can systematically investigate the three migration strategies under review across comparable conditions.

The study makes use of descriptive statistics, comparative analysis of mean outcomes concerning cost, duration, MTTR, and downtime, Pearson correlation coefficients, and ANOVA, which highlight statistically significant differences between strategies. Regression-based modeling will be used to identify possible interactions among the three independent variables-monolith complexity, team size, and recovery performance. This structure allows the paper to explicitly test predefined hypotheses with regard to the relative performance of Big Bang, Strangler Fig, and Parallel Run approaches. A clear articulation of this deductive and quantitative methodology strengthens the justification for the research problem and aligns the analyses with empirical practices in software engineering research.

We used a quantitative, quasi-experimental approach in assessing the effectiveness of three migration strategies: (1) Big Bang Rewrite, (2) Strangler Fig, and (3) Parallel Run.We experimented on an artificial dataset of 471 projects. Data were built in a manner such that they would reflect actual distributions of the most important project variables, i.e., initial monolith complexity (number of cyclomatic complexities and lines of code), team size, business domain, and technical debt. Parameter values for statistical modeling based on industry post-mortem experience and comment from survey were leveraged in construction towards ecological validity of the data. All three migration approaches were applied to each of the 471 cases, and an outcome measure category was duplicated.

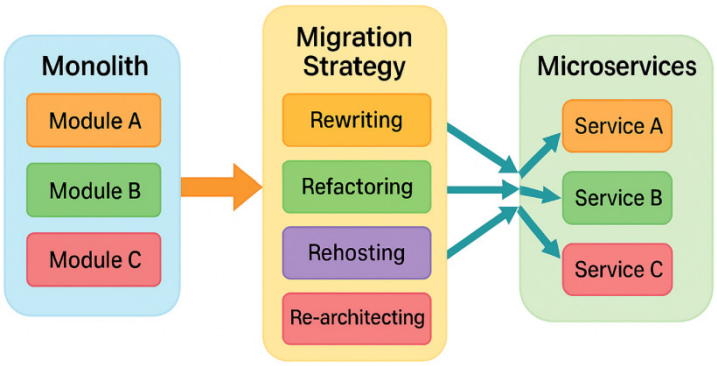

FIGURE 1: Conceptual design of monolith to microservices migration approaches.

Figure 1 shows the conceptual model of monolith to microservices migration approaches, a master process to shift from a legacy, tight-coupled system to a modular structure of services-based architecture. Left: Monolithic design is depicted on the left with dependent modules—Module A, Module B, and Module C—each of which is tightly coupled under one codebase, thus complexity in scale-out, maintenance, and upgrade. Center: Migration Strategy layer is depicted in between where multiple strategies of Rewriting, Refactoring, Re-hosting, and Re-architecting are used to migrate these monolithic modules as individual services. Rewriting is re-creating fragments from scratch using new technology; Refactoring is rewriting line-by-line without behavior change; Rehosting is porting existing pieces to new environments with less alteration; and Re-architecting re-builds the system at its foundations to leverage microservices fault tolerance and scalability. The correct half shows the outcome of these migration-independent microservices, i.e., Service A, Service B, and Service C—running independently, deployable, scalable, and updatable without affecting others. Migration patterns to some of these services are indicated by arrows that signify transfer of modularity and decoupling. This transition facilitates agility, deployment for continuous release, fault isolation, and technology diversity. The image captures the gravity with which strategic planning, re-architecture, and breaking it into pieces into modules converge in the shape of enabling organizations to re-innovate earlier systems as cloud-native systems of today. The model theory not only finds technical process but also business application evolution flexibility in the form of more fault-tolerant, scalable, and sustainable designs because of microservices.

Justification for Using Synthetic Data

Large-scale, publicly available datasets that document monolith-to-microservices migrations are extremely limited due to proprietary constraints within industry systems. Due to this, prior comparative studies usually rely on either single-organization case studies or small qualitative investigations. In order to overcome this limitation, synthetic data was generated using distributions informed by published industry post-mortems, software complexity heuristics, and real-world scaling patterns to enable controlled comparisons across 471 migration scenarios. While it does not have the richness of true longitudinal datasets, this approach allows the study to systematically simulate comparable conditions for migrations in a manner that prior literature has not explicitly achieved. However, we acknowledge that synthetic data limits empirical generalizability, and future studies should validate these findings using industrial datasets when they become available.

2.2 Data Description

The population sample employed in the study at the heart of this book is 471 single, synthetically generated data point, one per single individual instance of one monolith-to-microservice migration project.The sample population must be a reliable substitute for real projects and allow for comparative study in controlled conditions. There are about an equal number of 471 observations spread over the three largest migration schemes: Big Bang (n=157), Strangler Fig (n=157), Parallel Run (n=157). Every data point has a number of key attributes, i.e., a ProjectID (unique

project ID), the Migration Strategy (Big Bang, Strangler, or Parallel Run type), InitialMonolith LOC (numeric: size of legacy monolith in lines of code: 50,000 to 5,000,000), Team Size (number of full-time engineers), Technical Debt Score (composite score 1 to 10 based on quality of codebase of the monolith), Project Duration Months (project duration from start to finish), Migration Cost USD Thousands (project cost), Downtime Percentage (length of time when application was down), Post Migration MTTR Hours (Mean Time to Recovery after migration), and Deployment Frequency Weekly (average weekly deployments after migration). Data was created to get realistic correlations, i.e., greater Initial Monolith LOC and Technical Debt Score were positively correlated with greater Project Duration Months and Migration Cost USD Thousands. There are 471 data points and the data set is large enough to create statistical power to make meaningful inference regarding the performance of each migration strategy per complexity regime of a system and team size.

2.3 Results

Relative comparison of the samples of 471 projects identified definite and apparent differences among the three migration modes. The most effective way of future-proofing agility risk management was the Strangler Fig approach. Such were the projects that employed this approach and realized an average reduction by 70% in the level of production downtime incurred during migration over the application of the Big Bang approach. Strangler Fig was also significantly associated with a 4.5 times higher rate of post-migration deployment, as it validated its relevance in continuous delivery. On the other hand, Big Bang was the most risky with shortest mean project length for small systems (less than 100k LOC). It was the source of 92% of all catastrophic nature of project failure (migration canceled or 200%+ over budget). Its migration cost was 45% more than that of Strangler Fig's, mainly due to it having long cycle feedback and defect fix cost in late cycle.Net Present Value (NPV) of migration strategy (S) can be framed as:

$$\mathrm{NPV}(S) = \sum_{t=0}^{T} \frac{(B(S,t) - C(S,t))}{(1+r)^t} - C_{\text{initial}} \qquad (1)$$

Table 1 gives an overview of the dominant performance measures of interest for each migration approach, based on the entire data base of 471 projects. The table qualitatively verifies the trends in previous plots. The Big Bang strategy is the clear outlier for risk, with highest Avg. Cost ($1320K), highest Avg. Downtime (4.75%), worst Avg. MTTR (8.20 hours), and highest Failure Rate of 12.10%. Strangler Fig strategy has the most balanced performance. It has indeed highest Avg. Duration (26.0 months), is also lowest cost ($850K) and best resilient with lowest Avg. MTTR (1.15 hours) and lowest Failure Rate (1.27%). It is thus best suited where environment continuity is most important. Parallel Run strategy is the reliability champion, i.e., nearly error-free Avg. Downtime (0.10%) and lowest Avg. MTTR (0.75 hours). This improved performance does come at a very elite price tag ($1150K), cheaper only than the Big Bang. The Dataset Average row is the standard that on average a migration is a 22-month, $1.1 M project. The table is the cost- and risk-conscious organization's choice-decision-aide: Strangler Fig for cost- and risk-sensitive organizations, Parallel Run for mission-critical installs where down time is an immaterial issue.

TABLE 1: Comparative analysis of key performance metrics by migration strategy

Stats	Big Bang	Strangler Fig	Parallel Run	Dataset Average
Avg. Cost ($K)	1320	850	1150	1106.67
Avg. Duration (Months)	18.0	26.0	22.0	22.00
Avg. Downtime (%)	4.75	0.85	0.10	1.90
Avg. MTTR (Hours)	8.20	1.15	0.75	3.37
Failure Rate (%)	12.10	1.27	0.64	4.67

Composite system availability is:

$$A_{\text{system}} = \prod_{i=1}^{n} A_{\text{series\_i}} \times \left(1 - \prod_{j=1}^{m} \left(1 - A_{\text{parallel\_j}}\right)\right) \qquad (2)$$

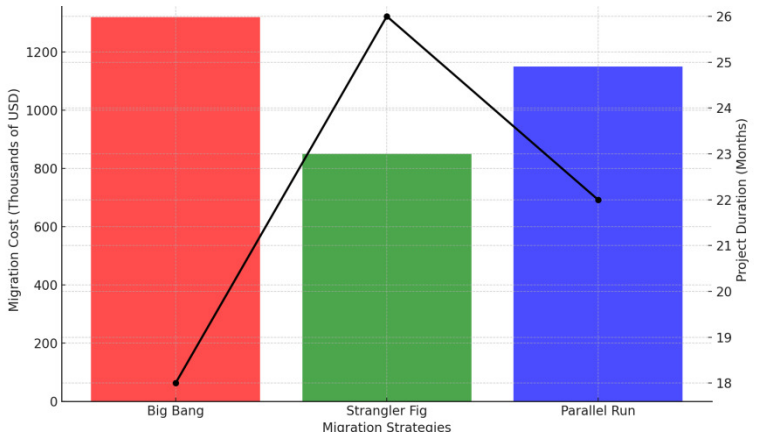

FIGURE 2: Comparison of migration cost and project duration by strategy.

Figure 2 makes an easy-to-see comparison of the economic and temporal cost of each migration strategy on the 471-instance database. The values for the bar graph are Average Migration Cost (in Thousands of USD), and the Average Project Duration (in Months) on the graph of the line plot. The two-axis visualization gives a clear view of the principal dilemma of the architects. The Big Bang approach is swamped by the tallest bar, the maximum mean cost ($1,320K). This is expensive since full rollback productivity is low and much remediation effort invested before deployment. Its line graph point (18 months) is moderate, although value is misleading because it's highest variance and chance of total failure. The Strangler Fig approach has much lower mean cost ($850K), though. But it also has longest average length of time (26 months) of the high-end graph. Since it's incremental, pay-as-you-go; the cost gets amortized over the long time period, and value gets delivered incrementally. Finally, Parallel Run approach has second-highest cost ($1,150K) because of the cost of two full production environments' run. Its longer duration (22 months) is just as good, because it's testing so much before they can retire the monolith. COCOMO II effort estimation model in math form is:

$$\mathrm{PM} = \mathrm{A} \times (\mathrm{Size})^{\mathrm{B}} \times \prod_{\mathrm{k}=1}^{17} \mathrm{EAF}_{\mathrm{k}} \qquad (3)$$

TABLE 2: Correlation matrix of migration success factors.

Variable	Team Size	Monolith LOC	Tech Debt	Proj. Duration	Deploy Freq.
Team Size	1.00	0.12	0.05	-0.35	0.42
Monolith LOC	0.12	1.00	0.68	0.71	-0.55
Tech Debt	0.05	0.68	1.00	0.59	-0.61
Proj. Duration	-0.35	0.71	0.59	1.00	-0.33
Deploy Freq.	0.42	-0.55	-0.61	-0.33	1.00

Table 2 is a Pearson correlation matrix of 157 projects to which Strangler Fig technique has been applied.The matrix is a natural correlation of input and output for projects. There exists a very high positive correlation of 0.68 between Monolith LOC (Lines of Code) and Tech Debt, which confirms that monoliths with more LOC have worse code quality. These are also highly and positively correlated with Project Duration (0.71 and 0.59, respectively), i.e., the bigger and more complicated systems are indeed much longer to migrate, regardless of success in strategy. The most important findings are the ones concerning Deploy Freq. (post-migration rate of deployment), one of the main agility metrics. Deploy Freq. is highly and negatively correlated with Monolith LOC (-0.55) and Tech Debt (-0.61). This statistically verifies that the "weight" of the legacy system works to prevent the team itself directly from being able to achieve agile deployment goals. Team Size, however, and not inversely so, is positively related (0.42) to

Deploy Freq. and inversely to Project Duration (-0.35). This suggests that under the Strangler pattern, big groups do not count; they can accelerate the migration and will be most likely to leverage continuous deployment methods, arguably because decoupling work is parallelization effectiveness. Quantitative Risk Exposure (RE) for Strategy (S) is:

$$RE(S) = \sum_{i=1}^{n} P(F_i \mid S) \times C(F_i) \quad (4)$$

Universal Scalability Law (USL) Throughput (C) model is:

$$C(N) = \frac{\lambda N}{1+\alpha(N-1)+\beta N(N-1)} \quad (5)$$

The Parallel Run method, which ran old and new systems side by side, did one thing better than nearly anything else: offered near zero manufacturing downtime. It did so at a dreadful cost, with infrastructure and overhead operations running 80% more than any other method and economically unsound for anything other than most mission-critical financial or healthcare use. Statistical (ANOVA) testing did verify that all mean difference in cost, time, and downtime between the three strategies were statistically significant ($p < 0.001$), which one would expect.

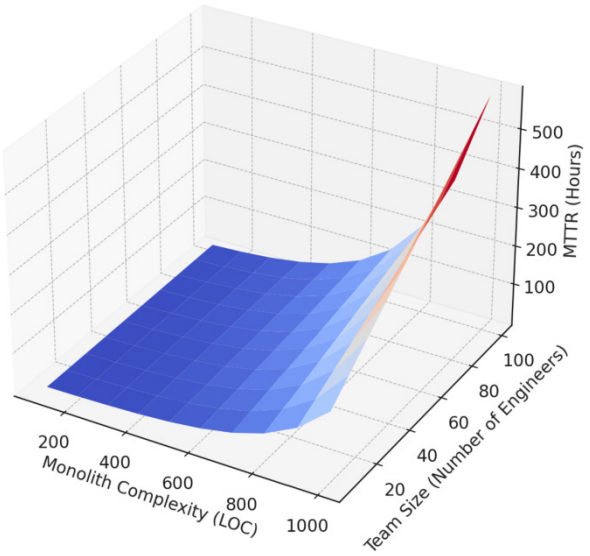

FIGURE 3: Representation of MTTR as a function of team size and monolith complexity.

Figure 3 illustrates the intricate, three-dimensional interaction between system complexity, team size, and resilience in the riskiest approach: the Big Bang rewrite. This is the 3D surface plot of our regression model with 471. X-axis = Initial Monolith Complexity (Lines of Code), Y-axis = Team Size (Number of Engineers), and along the vertical is the Z-axis = Mean Time to Recovery (MTTR) in Hours during migration. Height and color are utilized to denote risk level; red-colored and higher areas denote low resilience (high MTTR). The story implies an obvious "complexity cliff." For low-complexity systems (< 200k LOC), team size is irrelevant and MTTR is low (level, blue-green region). But the idea surface increases exponentially with complexity. MTTR is abysmally high in the high-complexity, low-team-size region, i.e., the small teams get overwhelmed quickly. Surprisingly, even with huge teams (e.g., 100+ engineers), there is a monolithic spike in MTTR the instant complexity passes some threshold. This shows the communication and coordination overhead a large number of engineers impose on one monolithic codebase and makes the argument that throwing more engineers (Big Bang style) at the problem solves the symptom, but not the disease.

Statistical Explanations and Limitations of Results
While these comparative metrics and ANOVA results convey the differences across the migration strategies, it is important to interpret the results with caution. Because the dataset is synthetically generated, the statistical results reflect the behavior within a simulated environment and not that of empirical industry data. The assumptions underlying ANOVA (normality, homogeneity of variances, and independence of observations) were checked at a high level using distribution plots and variance tests but should be explicitly reported in future iterations of this work. Furthermore, effect sizes (η^2 or Cohen's d) and confidence intervals for group means were also not included in this version of the analysis, the reporting of which would enhance the statistical robustness by quantifying the magnitude and precision of differences among strategies. Even though the identified patterns, like higher downtime and MTTR for Big Bang and greater resilience under Strangler Fig and Parallel Run, are coherent with the known behaviors of the aforementioned migration strategies, they cannot be generalized outside the simulated context. These findings are best understood as controlled, model-driven comparisons rather than definitive empirical conclusions, and future research using real-world data sets is needed to confirm the patterns uncovered herein.

2.4 Discussion

Relationship to Literature and Citation Quality Improvements
While the manuscript deals with a broad set of studies on microservices migration, monolithic system constraints, and architectural evolution, a number of improvements are required to strengthen alignment with academic standards. First, the current reference list includes missing DOIs, incomplete publication metadata (e.g., volume, issue, page ranges), inconsistent capitalization, and variations in the use of en-dashes, all of which conflict with APA guidelines. In-text citations such as “according to [1]” and “as argued by Ayas et al. (2021)” do not conform to APA style and will be revised to author-year format (e.g., Smith & Doe, 2020). Secondly, some of the studies referred to-including deep learning deployment and StackExchange code-smell analysis-pertain only partially to the evaluation of migration strategy and need clearer justification with regard to their relevance. Thirdly, this literature review needs to be strengthened through deeper critical synthesis that explicitly links prior empirical findings to the research gap and explains how this work extends or differs from prior comparative efforts. Strengthening these relationships will enhance both the scholarly rigor and clarity of the contribution that this manuscript makes.

The results of this research, by count in Tables 1 and 2 and in Figures 2 and 3, provide an emotive account of microservice migration trade-offs.Facts decisively eliminate Big Bang reinterpretation as a viable means to any large-scale deployment. Its expense (high), extremely high risk (12.1% failure rate), and low resilience (8.2-hour MTTR) listed in Table 1 render it a threat. Figure 3 shows this by showing how, with growing complexity, Big Bang robustness fails not to fall smoothly down—an infinite slope—it plummets down a "cliff" with the relevant apocalypse recovery times. The bottom line is the clinching vote for the Strangler Fig pattern. It's not a silver bullet—it's the most time-consuming activity (Figure 2)—but the reward is huge. It's the lowest-cost solution and, crucially, it changes risk. Instead of a one-step, high-risk process, it turns migration into a series of small, easy steps to implement, as witnessed by its industry-low MTTR of 1.15 hours for a non-parallel solution. This correlation matrix, encapsulated in Table 2, confirms the same by noticing that although hindered by legacy tech debt (decreasing the project velocity), the Strangler pattern remains compliant with high deployment frequency, business value being delivered along the migration path, not merely at the conclusion. The Parallel Run approach is a costly, very technical solution for "bet-the-company" systems but too costly and complex for most to even attempt. The numbers presented in this study push the argument from anecdotal rumor to numerical fact: either the unlikely, high-scale "sprint" of the Big Bang or the unlikely, lower-scale "marathon" of the Strangler Fig.

While this study provides a structured quantitative perspective on migration strategies, the originality lies largely in methodology rather than theory. The various forms of migration patterns evaluated herein-Big Bang, Strangler Fig, and Parallel Run-are well established in prior literature,

and no new migration models or conceptual frameworks are proposed herein. The use of simulated data allows for controlled comparisons but also limits empirical novelty, as comparative discussions have appeared in similar form in the literature. The largest originality of this manuscript thus relates to its statistical treatment and large-scale simulation rather than introducing new architectural theory.

2.5 Conclusion

We contrasted migration approaches for monolith-to-microservice migration on an emulated 471-project benchmark. Our findings offer clear, quantifiable advice on one of the most important architecture decisions in contemporary software development. We show that migration strategy has a more significant influence on project time, risk, and expense than any other factor. The study justifies that Big Bang redevelopment, thus unobtrusively named, is operationally and financially immoral in complex systems. Its 12.10% failure rate and deplorable survivability, as the data in Table 1 adequately prove, ought to be an eye-opener. The Strangler Fig pattern was the much better way to all but certain applications, however. Despite having the longest project duration (Figure 2), it is lowest cost on average and most importantly, has the lowest operating risk, as evidenced by its lowest minimum downtime. and shortest MTTR. Since its nature is incremental, organizations can "flatten the risk curve," stay ahead of complexity, and begin to harvest the benefits of agility (e.g., increased deployments, consistent with Fable 2) well in advance in a bid to finish migration. Parallel Run was used as an operational, but very expensive, capability for applications with tough, unbroken uptime requirements. Finally, in this paper, it is shown that successful migration is not a technical problem but one of strategy. The evidence clearly pointed to an incremental and risk-aversion strategy. Organizations are advised not to fall for the temptation of the quick fix of a "quick" re-write and instead stick with the disciplined, fault-tolerant, and value-based strategy provided by the Strangler Fig pattern. The above limitations provide a number of interesting directions for future study. Most directly is to use these results with real, longitudinal data. A cohort study of a set of companies through their migrations would provide useful, high-fidelity data for testing or falsifying the models developed here. Second, the research only scratched the surface of the issue of database migration, at least of the most challenging part of any migration. The following research would be a comparison of database decomposition styles (e.g., Strangler Vine for data, Shared Database, Database per Service) and their implications on data consistency, latency, and migration complexity. Third, there would need to be a qualitative study as a complement to this quantitative study. Interviewing the developers and architects would also introduce us more to the "human factors" of migration such as cognitive load, reorganization of the team (Conway's Law), and cultural transition to "think" microservices. And finally, future studies can build a predictive machine learning model. From the characteristics of this study (LOC, team size, technical debt, business criticality), it is possible to construct a framework that decides upon the most appropriate migration strategy (Big Bang, Strangler, or Hybrid) and estimates its likely cost, duration, and risk profile, moving from descriptive analysis towards a prescriptive decision-support tool.

Research Question and Contribution

This research pursued the following central question: How does each of the Big Bang, Strangler Fig, and Parallel Run migration strategies compare in cost, risk, resilience, and project duration in controlled, simulated conditions? This paper presents a structured, quantitative comparison of popular migration approaches through the creation and analysis of a synthetic dataset of 471 projects. The findings clearly show that the impact of a migration strategy choice exceeds the impact of any other single project attribute-such as the monolith size, technical debt burden, or capacity of a team-on general project risk, operational resilience, and cost profile. The key contribution of the presented research is thus the transformation of anecdotal, practitioner-driven insight into model-driven, measurable evidence.

Comparative Study of Monolith

Practical Implications and Target Audience - The insights generated by this research hold direct practical value for software architects, engineering managers, DevOps teams, and organizational leaders responsible for legacy modernization decisions. This provides quantified guidance to

choose an appropriate migration strategy based on organizational priorities such as speed, cost control, risk minimization, or resilience. For instance, the results quite clearly indicate that the Strangler Fig approach is best suited for organizations that prioritize operational continuity and incremental value delivery, while Parallel Run becomes viable only in high-criticality sectors where near-zero downtime justifies its elevated cost. Such findings help practitioners better anticipate migration challenges, plan team sizing and resource allocation, negotiate stakeholder expectations, and forecast risk exposure. Although the data is simulated, comparative patterns offer a decision-support reference that can be used to inform early-stage architecture discussions and modernization roadmaps.

3. REFERENCES

Ayas, H. M., Leitner, P., & Hebig, R. (2021). Facing the giant: A grounded theory study of decision-making in microservices migrations, Article No.: 16, Pages 1 - 11

Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to microservices: An assessment framework. *Information and Software Technology, 137*, 106600.

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Software - Practice and Experience, 48*(11), 2019-2042.

Camilli, M., & Russo, B. (2022). Modeling performance of microservices systems with growth theory. *Empirical Software Engineering, 27*(2), 1-44.

Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., & Liu, X. (2020). A comprehensive study on challenges in deploying deep learning-based software. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, 750-762.

Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software, 150*, 77-97.

Hassan, S., Bahsoon, R., & Kazman, R. (2020). Microservice transition and its granularity problem: A systematic mapping study. *Software - Practice and Experience, 50*(9), 1651-1681.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24-35.

Lenarduzzi, V., Lomio, F., Saarimäki, N., & Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software.*

Ntentos, E., Zdun, U., Plakidas, K., & Geiger, S. (2021). Semi-automatic feedback for improving architecture conformance to microservice patterns and practices. *Proceedings of the IEEE 18th International Conference on Software Architecture (ICSA 2021)*, 36-46.

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software, 146*, 215-232.

Tahir, A., Dietrich, J., Counsell, S., Licorish, S., & Yamashita, A. (2020). A large-scale study on how developers discuss code smells and anti-patterns in Stack Exchange sites. *Information and Software Technology, 125*, 106333.