PAEEM: Phase-Aware Effort Estimation Model for Effort Forecasting in SOA-to-MSA Migration

Sandeep Sharma[1], Vijay Pal Singh[2]
[1,2]Department, of Computer Engineering & Applications, Mangalayatan University, Aligarh, India

Abstract: A gradual transition from Service-Oriented Architecture (SOA) to Microservices Architecture (MSA) is increasingly adopted to minimise disruption during digital transformation. However, the estimation of the engineering effort for each phase of the migration is challenging due to service dependencies, rollback risk, and team maturity. This paper presents PAEEM (Phase-Aware Effort Estimation Model), a lightweight structured model for estimating phase-wise migration effort in SOA-to-MSA transitions. PAEEM consists of three measures: the Phase Overlap Index (POI) that quantifies inflation of effort due to inter-phase service coupling; the Rollback Complexity Index (RCI) that quantifies risk in the deployment, testing and data dependencies; and the Learning Curve Factor (LCF) that characterised the efficiency gains in consecutive migration phases. By combining architectural analysis with expert judgement, the model is well suited to early-stage planning and low-tooling. A simulated multi-phase migration case study of an e-commerce system is used to demonstrate PAEEM's effectiveness, yielding a Mean Absolute Percentage Error (MAPE) of 8.13%, significantly outperforming reference models. The model provides a convenient, flexible approach to researchers and practitioners involved in planning incremental application modernisation from SOA to MSA.

Keywords: SOA, Microservices Architecture (MSA), Phase-Aware Effort Estimation Model (PAEEM), Phase Overlap Index (POI), Rollback Complexity Index (RCI), Learning Curve Factor (LCF), Application Modernisation.

1. Introduction

The shift from Service-Oriented Architecture (SOA) to Microservices Architecture (MSA) is now a typical enterprise system modernisation policy. Phased migration is a popular technique in practice among other existing methods such as the strangler pattern, event-first, and domain-driven architectures, since it allows organizations to reduce disruption as deployment strategies are incrementally tested [1]. Phased migration, however, creates a challenging planning problem like how well to estimate the engineering effort to get through each migration phase. Current models of effort estimation (COCOMO II, COSYSMO, WMFP, and even the latest models relying on machine

learning, e.g. SE^{3}M) are optimised for monolithic system level or project level estimation. These models do not include constructs to model incremental migration phases, dependencies between services or the evolving maturity of engineering teams [2].

Empirical research and architecture recovery tools solely focus on feasibility or restructuring, without empirically estimating the engineering effort per phase [3]. Furthermore, contextual specific issues, including service coupling, rollback risks, and learning progression, distribute the effort disproportionately across phases, and the traditional estimation methods fail to capture this variation [4].

In this paper, we present the Phase-Aware Effort Estimation Model (PAEEM), a simple but organised model specific phase-wise SOA-to-MSA migration. PAEEM introduces three measures which can be computed in the form of an index: Phase Overlap Index (POI), Rollback Complexity Index (RCI) and Learning Curve Factor (LCF), which are determined by human judgement and architecture analysis. These metrics can be evaluated without historical data or automated tools, making the model useful in early-stage planning and resource-constrained settings [5].

A simulated case study validates PAEEM: it outperforms common heuristics and keeps error low with a MAPE only 8.13%. This paper makes the following contributions:

- A novel model for estimating engineering effort during phased SOA-to-MSA migrations.
- Three context-aware metrics, POI, RCI, and LCF addresses service coupling, operational risk, and learning progression.
- A simulation-based evaluation showing improved accuracy over baseline estimation methods.

Figure 1 shows the main contributions of PAEEM: its migration model, context-aware metrics (POI, RCI, LCF), and validation through simulation.

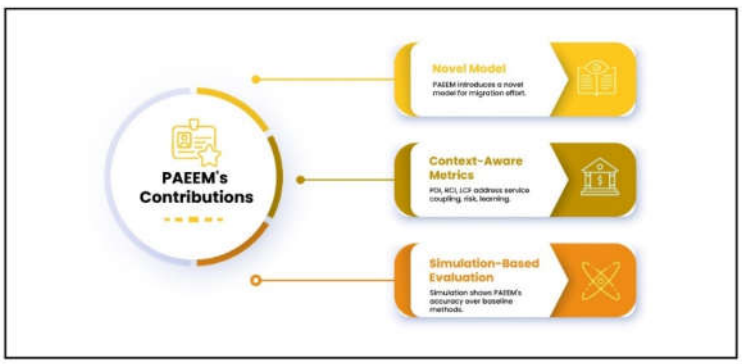

Figure 1. Contributions of PAEEM

2. Related work

Estimation of effort is still a component of software project planning, but the majority of existing models assume monolithic or homogeneous development cycles, rather than incremental architectural change [6]. COCOMO II, COSYSMO, WMFP, and SE3M are widely used as general-purpose estimation models based on either size or data-driven learning. However, they lack constructs to model effort in multi-phase migrations, such as SOA-to-MSA transitions [7].

COCOMO II has effort as a scale/cost modulated function of system size (e.g. KLOC or function points) [8]. It is empirically sound, but it views entire delivery structure as a monolith and does not consider the compounded effort of inter-phase dependencies or rollback risks of a staged migration. COSYSMO generalises this reasoning to systems engineering by adding counts of interfaces and scenarios but again abstracts the dynamics of staged deployment and inter-service coordination common in the adoption of microservices [9].

Weighted Micro Function Points (WMFP) is a code perspective, an artifact-based source estimator of effort. It is not dynamic and captures neither changing runtime behaviours nor data transformations or gradual refactoring, which are elements of migration work that are important in scoping legacy systems [10]. SE^3M is a text-based estimating tool, which employs pre-trained textual requirement embeddings to estimate effort. SE^3M is a black box and has limited interpretability and relevance in migration settings with sparse historical data density and heterogeneous architectural complexity, albeit flexible and adaptive on a large scale [11].

Simultaneously, an increasingly rich empirical literature on migration offers understanding of applied issues related to SOA-to-MSA transformations. They include synchronous dependencies management, shared databases, absence of rollback strategy, and blue/green deployment model or canary deployment model. These studies are useful in identifying patterns and recording risks but are mainly qualitative like they seldom offer quantitative models of the phase-dependent estimation of effort [12].

The Phase-Aware Effort Estimation Model (PAEEM) is a simple, yet formalised model of effort estimation during staged migrations. In contrast to data-intensive models or models that are not dynamic, PAEEM presents three metrics that can be scored manually: the Phase Overlap Index (POI) which measures service dependencies, the Rollback Complexity Index (RCI) that measures test and deployment risk and the Learning Curve Factor (LCF) to indicate growing team efficiency. These elements combined allow interpretable, recalculable, and context-sensitive estimation with no historical data or automation tooling. PAEEM can be especially applied in situations where the population of interest is small, the research is academic and where resources are limited and intent is to align estimation activities with practical realities of migration**.**

PAEEM satisfies a critical gap in the estimation literature by instantiating architectural dependencies, operational risks, and organizational learning into its framework, providing a phase-sensitive, migration-specific alternative to both classical models and anecdotal practitioner wisdom [13].

Table 1 presents the critical literature gaps, their significance and shows how PAEEM addresses them through phase-specific, interpretable, and applicable metrics.

Table 1. Summary of Literature Gaps and PAEEM Contributions

Sr. No.	**Gap in Existing Models/Studies**	**Significance of the Gap**	**How PAEEM Addresses It**
1	No phase-wise effort estimation in traditional models (COCOMO II, COSYSMO, WMFP, SE^{3}M)	Migration happens in phases, each with unique dependencies and risks	Provides per-phase effort via a structured formula
2	No quantification of service coupling across phases	Cross-phase calls/data-sharing inflate integration/testing cost	Introducing Phase Overlap Index (POI)
3	Rollback/test/data risks not included in estimates	Critical for safe migration, especially in high-risk domains	Adds Rollback Complexity Index (RCI)
4	No modelling of team learning/improvement over phases	Teams gain efficiency, reducing effort in later phases	Models via Learning Curve Factor (LCF)

5	Heavy reliance on historical data or automation tools	Limits used in new projects, academia, or resource-constrained teams	Fully manual, transparent, and lightweight
6	Empirical studies lack actionable estimators	Offer useful observations, but no phase-aligned predictions	Converts qualitative risks into effort deltas
7	Poor interpretability in ML-based models (e.g., SE^3M)	Black-box predictions hinder trust and adaptation	Human-readable formula with expert-tunable weights

PAEEM is a step toward reconciliation between theory-based models of effort and realistic risk-sensitive models of migration. It enables realistic estimation even when available tooling is limited. Thus, it is the model that suits the situation of academic research, agile teams and digital transformation.

3. Methodology

In this section, the systematic procedure employed to develop the Phase-Aware Effort Estimation Model (PAEEM) is described. The approach aims to enable an organised and manual estimation of effort in each phase of an incremental migration from SOA to MSA, without the assistance of automation or big data.

The process includes phase identification, metrics formulation, and application of a weighted estimation model. All the components are evaluated manually using architectural understanding, professional input, and logical grouping principles.

3.1. Model Overview

The proposed Phase-Aware Effort Estimation Model (PAEEM) has enabled rigorous engineering effort estimation by providing the estimation during the incremental migration from Service-Oriented Architecture (SOA) to Microservice Architecture (MSA). The model acknowledges that migration is never unidimensional or homogenised. It also acknowledges that it treats each of the phases as an independent unit of effort subject to architectural dependencies, operational risks, and

organizational learning. Because PAEEM is a manually driven model which is easy to apply and interpret, it may be applicable to both academic and small-enterprise settings where automated tools are not available.

3.2. Phase Identification and Service Grouping

PAEEM is based on the determination of logical groupings of services (phases) for migration. Services are clustered based on service coupling, business-domain alignment, operational risk, and team readiness. This is achieved by manual architectural analysis (e.g. service-dependency diagrams, business-process mappings) and human judgement.

Determination of the phases is done by the following steps:

- List all the SOA services in the legacy system.
- Map service dependencies to interpret about coupling and cohesion.
- Group services that are tightly integrated or share business related logic.

Step through the order of the lowest risk, lowest complexity services to more important and tightly coupled modules. Effort is estimated in each phase as an independent unit but may depend on the properties of prior phases.

3.3. Metrics

PAEEM proposes three fundamental metrics that are used at every phase of migration. The metrics are evaluated manually through architectural and operational evaluation.

3.3.1. Baseline Phase Effort (BPE)

Baseline effort is estimated at the task level within each story. Each task t is assigned a base hands-on duration, h(t) in hours (Effective Engineering Hours, EEH). No separate complexity multiplier is used. The implementation difficulty is captured directly in $h(t)$. Migration-specific dependency/rollback risk and team learning are applied later. These are determined at the phase level through different phase-level metrics.

$$\mathbf{BPE_{phase(p)}} = \sum_{m \in \mathbf{M_p}} \sum_{s \in \mathbf{S_m}} \sum_{t \in \mathbf{T_s}} \mathbf{h(t)}$$

where:

- $\mathbf{M_p}$ is the microservices delivered in phase p.
- $\mathbf{S_m}$ is the stories belonging to microservice m (in phase p).
- $\mathbf{T_s}$ is the task inside the story s.
- **h(t)** is the effective engineering effort (EEH) to complete the task.

Each task t is assigned a base hands-on duration $h(t)$ in **Effective Engineering Hours (EEH)** using a fixed grid. **h(t)∈ {0.5,1,2,3,4,6,8}** hours.

- If a task > 8 h, split it.
- If < 0.5 h, merge with another tiny task.

3.3.2. Phase Overlap Index (POI)

POI measures the degree of service coupling between the current phase and those microservices that have already been migrated in the previous phases. The higher the POI, the more dependency and therefore more integration and coordination work is required.

$$\mathbf{POI}_p = \frac{\mathbf{C}_p}{\mathbf{T}_p} \quad , \quad \mathbf{0 \leq POI}_p \leq \mathbf{1}$$

Where:

- $\mathbf{C_p}$ is the number of microservices in the current release that have upstream dependencies on earlier-phases services (API calls, shared datasets, libraries, or other couplings).
- $\mathbf{T_p}$ is the total number of microservices in the current stage.

POI can have a minimum value of 0.0 (completely independent) and a maximum value of 1.0 (completely dependent), practically, a POI value greater than 0.5 means that the integration is very complex.

3.3.3. Rollback Complexity Index (RCI)

RCI is used to estimate the risk of testing, deploying or rollback of a phase. It is based on a list of risk factors such as shared data storage, manual deployment, and absence of rollback strategies.

$$\mathbf{RCI}_p = \frac{\mathbf{R_p}}{\mathbf{N}} \quad , \quad \mathbf{0 \leq RCI}_p \leq \mathbf{1}$$

Where:

- $\mathbf{R_p}$ is the number of rollback risks that have been identified in the current phase.
- **N** is the total number of risk factors. In this study, N = 10 is used; other groups can choose another N.

Common risk factors include shared databases, inadequate test automation, brittle legacy dependencies, and the absence of defined rollback plans. The RCI values range from 0.0 (no risk) to 1.0 (high risk).

Table 2 lists the predefined risk factors used in this study; each item is checked per phase to compute the Rollback Complexity Index (RCI).

Table 2. Risk factor and description

Sr. No.	Risk Factor	Description
1	Shared Database Dependencies	Service in this phase accesses a database shared with non-migrated components, complicating rollback and data consistency [14], [15], [16].
2	Improper Service Granularity	Services being migrated are over coarse or over fine-grained, increasing refactoring and testing complexity [17, 18].
3	Lack of API Versioning	Public interfaces lack proper version control, making rollback unsafe or disruptive to consumers [15], [19].
4	Distributed Transaction Handling	The phase includes workflows involving transactions across multiple services, increasing coupling and rollback risk [14], [16], [18]

5	Incomplete Rollback Strategy	No defined or tested rollback process exists for this phase, making recovery difficult after failure [16], [20].
6	No Automated Testing Framework	Phase lacks sufficient unit, integration, or regression test coverage to safely validate the migration [16], [18], [20].
7	Environment Configuration Drift	Inconsistent development, testing or production environments lead to failures during rollback or redeployment [15], [16], [19].
8	Dependency on Legacy Middleware	Reliance on old ESBs or SOAP-based middleware adds fragility during rollback attempts [17-19].
9	Insufficient Monitoring & Alerts	Lack of observability makes it difficult to detect failures early, delaying rollback actions [14], [15], [18].
10	Inadequate CI/CD Rollback Support	Pipelines do not support "one-click" rollback (blue/green, canary), making reversals slow and error-prone [16], [17], [20].

3.3.4. Learning Curve Factor (LCF)

LCF represents productivity knowledge gains of the team with each successive migration phase. Later on, as experience builds, the teams will work more efficiently, and team requires less effort in next phases.

$$\mathbf{LCF}_p = \mathbf{min}\left(\left(\mathbf{p}-\mathbf{1}\right)\times \mathbf{G}, \mathbf{0.3}\right), \qquad \mathbf{0} \le \mathbf{LCF}_p \le \mathbf{0.3}$$

Where:

- **p** is the phase index (p = 1 ,2, …).
- **G** is the gain of a per-phase that has been selected by the expert panel (e.g., G = 0.1).

LCF can only take a limit of 0.3 to prevent over discounting.

To determine the value of G, PAEEM uses a structured capability checklist covering five maturity factors that impact team efficiency. These factors are listed in Table 3 and must be evaluated in each migration phase.

Table 3. Capability Checklist for Learning Curve Factor (LCF)

Sr. No.	Capability (Quality Check item)	Short description	Minimal evidence to mark Yes
1	Service template / generator	Standardised scaffold ensures new services start with uniform logging, config, health checks, and CI hooks.	Template repository or consistent pull request usage
2	Reusable CI/CD + shared IaC/Helm	Common pipelines and infrastructure modules reused for build, test, deployment, and environment provisioning.	Reusable workflow (GitHub Actions/Jenkins) **and** shared Helm/Terraform modules referenced by service repos.
3	Reusable test harness with CI gates	Ready unit, integration, and contract tests integrated into CI with quality gates to prevent regressions.	CI logs show test jobs and coverage/contract gates passing.
4	Golden-path docs / runbooks	Curated playbooks for build, deployment, rollback, and observability, linked where developers work.	Links in README/PR templates; runbooks version controlled.
5	Data migration & cutover checklist	Versioned schemas, backfill/dual-write scripts, and a tested cutover/rollback procedure.	Migration scripts + schema versioning in Version Control System; evidence of dry run/cutover test.

The number of "Yes" answers from this checklist Quality Checklist (QC) score to determines the learning gain value G, which is mapped using the predefined scale shown in Table 4.

Table 4. Gain(G) finder matrix

Quality checklist (QC)	0	1	2	3	4	5
G (per-phase gain)	0	0.02	0.04	0.06	0.08	0.1

Values were calibrated based on consensus of three practitioners who had executed SOA-to-MSA migrations. The non-linear increments reflect diminishing returns at lower maturity and accelerated gains once teams adopt advanced practices (QC ≥3).

3.3.5. Phase Modifier (PM)

PM scales the baseline story-point estimate of a migration phase based on two normalised drivers:

- Cross-phase overlap (POI_{norm})
- Rollback/operational risk (RCI_{norm}).

Table 5 presents a matrix that provides the Phase Modifier (PM) as a look-up over normalised Rollback Complexity Index (RCI_{norm}) and Phase Overlap Index (POI_{norm})

Table 5. PM finder matrix

RCI_{norm} ↓ \ POI_{norm} →	0–0.20	0.20–0.40	0.40–0.60	0.60–0.80	0.80–1.00
0–0.20	1.06	1.14	1.22	1.3	1.38
0.20–0.40	1.1	1.18	1.26	1.34	1.42
0.40–0.60	1.14	1.22	1.3	1.38	1.46
0.60–0.80	1.18	1.26	1.34	1.42	1.5
0.80–1.00	1.22	1.3	1.38	1.46	1.54

Note: Band boundaries are inclusive on the left; for example, a value of 0.20 falls in the 0.20–0.40 band, not the 0–0.20 band.

The PM values in table 5 were manually calibrated using input from three practitioners with hands-on SOA-to-MSA migration experience. The resulting ranges align with enterprise phased migrations, where architectural coupling and operational risk increase the required effort.

Calibration ensures minimal dependencies and rollback risk add overhead (PM ≥ 1.06), while the maximum observed phase-level effort phases does not exceed ~54%. This bounded range reflects expert consensus on practical limits in enterprise SOA-to-MSA migrations.

3.4. Phase-Aware Effort Estimation Model (PAEEM)

PAEEM combines the three metrics mentioned earlier, POI, RCI, and LCF into a weighted equation to measure the engineering effort needed at each phase of migration. The approximate effort of a certain phase is calculated as:

$$\mathbf{Effort_{(p)}} = \mathbf{BPE_p} \times \mathbf{PM_p}(\mathbf{POI_{norm}}, \mathbf{RCI_{norm}}) \times (\mathbf{1} - \mathbf{LCF_p})$$

Where:

- $\mathbf{BPE_p}$ is the baseline effort (EEH) in phase p.
- $\mathbf{PM_p}$ is the phase modifier of phase p based on POI and RCI, shown in table 5**.**

POI and RCI normalised in [0,1] and LCF normalised in [0,0.3] because of cap. This model ensures that the engineering effort grows in proportion to technical dependencies and rollback risk and offers the opportunity to reduce it depending on team learning and process maturity in successive phases.

This methodology defines a structured, human-evaluable pipeline for estimating phase-wise effort in SOA-to-MSA transitions. The following section presents a simulated application of PAEEM and a comparison with standard estimation models.

4. Application and Simulation of PAEEM

4.1. System Description

To illustrate the applicability of PAEEM, we model an e-commerce system that was initially developed as a service-oriented architecture (SOA) of four coarse-grained services: User, Product, Payment, Order that communicate over XML/SOAP-style interfaces. Despite its utility, this coarse-grained SOA limits scale, deployability, and agility, and prompted a microservices (MSA) migration.

4.2. Migration Phases

This system was gradually broken down into ten microservices and migrated over four phases in a domain-driven approach to isolate business capabilities, minimise inter-phase dependencies, and align with team readiness.

Table 6. Migration Phases and Microservice Allocation

Phase	**Microservices Migrated**
P1	Authorization, Profile, Session Management
P2	Catalog, Search
P3	Review & Rating, Cart, Checkout
P4	Invoice, PaymentGateway

This staged design scopes and verifies each of these steps individually while ensuring business continuity is in place.

4.3. Metric Scoring (POI/RCI/LCF)

At each stage, the three PAEEM metrics, POI, RCI, and LCF, are rated by architectural review, simulated risk scenarios, and a structured learning sequence.

Table 7 presents the normalised phase-level scores: Phase Overlap Index (POI), Rollback Complexity Index (RCI), and Learning Curve Factor (LCF) derived from G. These values serve as direct inputs to the Phase Effort Model (Table 9)

Table 7. Metrics Inputs and Scoring per Phase (POI, RCI, LCF).

Phase (P)	**C**	**T_p**	**POI= C/T_p**	**R**	**N**	**RCI = R/N**	**G**	**LCF = (p−1) × G**
P1	0	3	0	3	10	0.3	0	0
P2	2	2	1	2	10	0.2	0.04	0.04
P3	2	3	0.67	4	10	0.4	0.06	0.12
P4	1	2	0.5	5	10	0.5	0.08	0.24

4.4. Per-Service Baseline & Phase Effort

The per-service baseline and phase effort for each phase is mentioned below in Table 8.

Table 8. Per-Service Baseline engineering efforts (EEH) and Phase Totals

Phase	Microservice	Number of Story	0.5-h Tasks	1-h Tasks	2-h Tasks	3-h Tasks	4-h Tasks	6-h Tasks	8-h Tasks	Total EEH (hours)	Baseline Phase Effort (BPE) in EEH
P1	Authorization	10	0	0	2	2	4	3	4	**76**	**133**
	Profile	6	0	1	0	2	2	0	2	**31**	
	Session Mgmt	5	0	0	1	0	2	0	2	**26**	
P2	Catalog	9	0	2	0	0	2	3	3	**52**	**89**
	Search	7	0	2	0	1	2	0	3	**37**	
P3	Review & Rating	7	0	0	0	2	2	2	3	**50**	**197**
	Cart	8	0	0	1	1	3	4	4	**73**	
	Checkout	9	0	0	0	2	3	4	4	**74**	
P4	Invoice	6	0	0	1	2	2	2	3	**52**	**194**
	PaymentGateway	10	0	1	0	3	5	8	8	**142**	

Using the BPE values from Table 8 and the metric scores from Table 7, the effort for each phase is calculated as:

$$\mathbf{Effort_{(p)}} = \mathbf{BPE_p} \times \mathbf{PM_p}(\mathbf{POI_{norm}}, \mathbf{RCI_{norm}}) \times (1 - \mathbf{LCF_p})$$

Table 9. PAEEM Effort per Phase

Phase	BPE	POI	RCI	LCF	PM	Effort phase (EEH)
P1	133	0	0.3	0	1.1	146.30

P2	89	1	0.2	0.04	1.42	121.32
P3	197	0.67	0.4	0.12	1.38	239.24
P4	194	0.5	0.5	0.24	1.3	191.67

5. Results

5.1. Evaluation Design and Baselines

In order to determine the practical accuracy and contextual relevance of the Phase-Aware Effort Estimation Model (PAEEM), the model's predictions were evaluated against three baseline approaches commonly used in migration planning. Uniform Effort Distribution Model assumes an equal effort distribution across all phases (e.g., 100 story points per phase). Meanwhile, Service Count Model allocates effort proportionally based on the number of microservices migrated per phase. These models are useful as they are simple, but none of them consider inter-service dependencies, rollback risks, or learning as a team over time.

COCOMO II, COSYSMO, and SE^{3}M (and many more) are project-level or monolithic software estimation models that use static parameters (e.g., KLOC, function points, or historical corpora). These models do not allow per-phase estimation of effort or migration-related constructs such as inter-phase dependencies or rollback risk. In this light, a direct comparison would be structurally invalid and misleading. Rather, we compare PAEEM with two realistic heuristics, uniform distribution of effort and service-count based estimation, which were widely used in phased migration planning.

Table 10. Summarises the estimation outcomes across the four migration phases:

Phase	Microservices	Actual Efforts (EEH)	Uniform Effort Model (EEH)	Service-Count Model (EEH)	PAEEM (EEH)
P1	3	168	153.25	183.9	146.30
P2	2	124	153.25	122.6	121.32
P3	3	218	153.25	183.9	239.24
P4	2	178	153.25	122.6	191.67

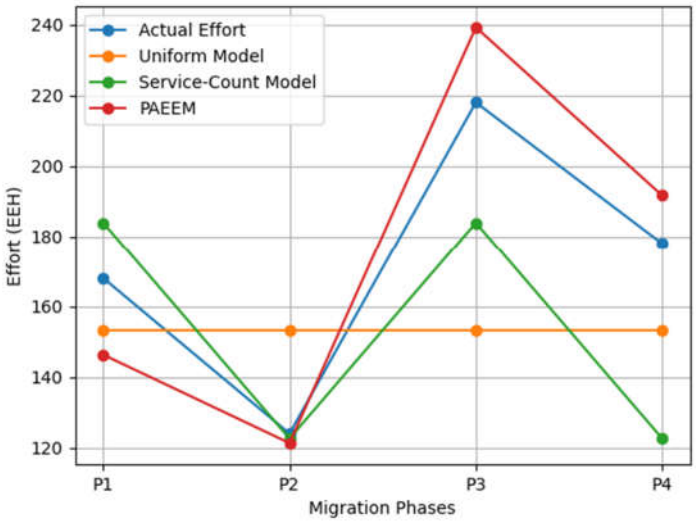

Figure 2.Phase-wise comparison of actual and estimated effort, highlighting PAEEM's improved accuracy.

Systematic expert judgement was used to determine the actual effort values due to the complexity of services, integration requirements, and readiness to deploy.

5.2. Accuracy Analysis

To quantify estimation performance, the Mean Absolute Percentage Error (MAPE) is employed:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{(\text{Estimated}_i - \text{Actual}_i)}{\text{Actual}_i}\right| * 100$$

Table 11. Aggregate Accuracy (MAPE)

Model	MAPE (%)
PAEEM	8.13
Service-Count	14.34
Uniform Effort	18.99

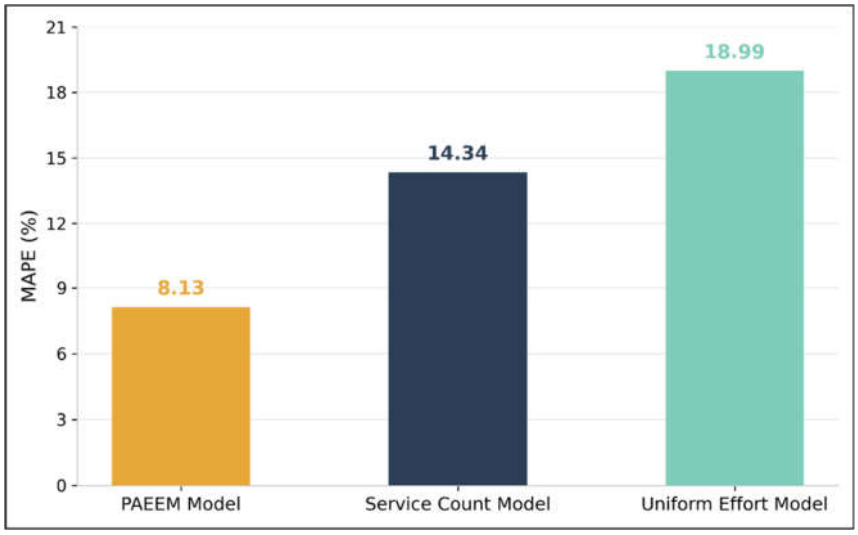

Figure 3. Accuracy comparison between models

As shown in Table 11 and Figure 3, PAEEM achieves a significantly lower error rate, which highlights its ability to capture context-specific variables such as cross-phase service dependencies, rollback complexity, and learning curve effects that are typically overlooked by traditional heuristics.

6. Discussion

6.1. Interpretation of Findings

PAEEM has achieved better accuracy by tuning the base effort using context-aware modifiers. The Phase Modifier (PM) integrates the Phase Overlap Index (POI) and Rollback Complexity Index (RCI) to control architectural dependencies and operational risk in each phase. The Learning Curve Factor (LCF) reduces effort estimates to reflect team efficiency gains over time. Equal Effort and Service Count models assume static effort distributions while PAEEM dynamically scales effort estimates based to the specific complexity of each phase. This approach resonates closely with actual engineering effort, particularly during phases with strong coupling or risk. The interpretable structure and manual scoring process make model well suited for early-stage planning and settings even when historical data or tools are limited.

6.2. Limitation

Construct-wise, effort values were normalised to EEH, which may not capture team-specific estimation styles. Internally, the metrics scores (POI, RCI, LCF) were based on expert judgement,

introducing possible subjectivity. Meanwhile externally, the evaluation was limited to one simulated system, affecting generalizability. The small number of migration phases (n=4) restricts statistical inference. While the model is fully documented, adaptations may be necessary for use in other settings.

6.3. Future Work

Future work will focus on validation of model on real migration projects in different domains, by extending to cost estimation, scheduling, and capability for enterprise-scale planning. This will include integrating lightweight analytics to support manual scoring and building hybrid estimators that combine expert knowledge with data-driven learning. By tightening metrics calibration through broader expert panels or learning-based adjustments. With further refinement and rigorous validation, PAEEM can develop into a practical and adaptable estimator for modern software-migration programs.

7. Conclusion

This paper has introduced PAEEM (Phase-Aware Effort Estimation Model), a simple, lightweight and easy-to-understand model to estimate engineering efforts during incremental migration from SOA to MSA. There are three key migration-specific factors that differentiate PAEEM from traditional models; Phase Overlap Index (POI) which establishes architectural coupling, Rollback Complexity Index (RCI) which establishes the level of deployment risk and Learning Curve Factor (LCF) establishes team maturity with the new procedure. In a controlled case study, the model achieved a MAPE of 8.13%, outperforming both the Service-Count model (14.34%) and the Uniform Effort model (18.99%), demonstrating that it consistently estimated efforts more accurately than the baseline methods. This makes PAEEM a practical choice for planning during early stages, in academic research contexts, and in low-resource environments without losing clarity. PAEEM has shown effective results, but it still depends on expert judgement and simulated tests, which limits its use in large, real-world programs.

References

[1] B. S. Beeraka, “Modernizing enterprise architecture: A guide to microservices migration and hybrid cloud integration,” *International Journal of Science and Research Archive*, vol. 14, no. 1, pp. 440–447, Jan. 2025, doi: https://doi.org/10.30574/ijsra.2025.14.1.0052.

[2] Vinay Raj and R. Sadam, “Patterns for Migration of SOA Based Applications to Microservices Architecture,” *Journal of Web Engineering*, Jul. 2021, doi: https://doi.org/10.13052/jwe1540-9589.2051.

[3] D. M. Singh and D. R. Tripathi, “TRADITIONAL AND RECENT TECHNIQUES OF EFFORT ESTIMATION IN SOFTWARE ENGINEERING,” *BSSS Journal of Computer*, vol. 14, no. 1, pp. 11–21, Jun. 2023, doi: https://doi.org/10.51767/jc1403.

[4] H. Chen, B. Xu, and K. Zhong, “Enhancing Software Effort Estimation through Reinforcement Learning-based Project Management-Oriented Feature Selection,” *arXiv (Cornell University)*, Mar. 2024, doi: https://doi.org/10.48550/arxiv.2403.16749.

[5] Imam Habib Pamungkas, None Suhardi, and Sawung Murdha Anggara, “Identifying Readiness Criteria for Architecture Migration from Monolithic to Microservices: A Literature Review,” *2024 International Conference on ICT for Smart Society (ICISS)*, pp. 1–6, Sep. 2024, doi: https://doi.org/10.1109/iciss62896.2024.10751305.

[6] M. Shepperd, “‘Estimating Software Project Effort Using Analogies’: Reflections After 28 Years,” *IEEE Transactions on Software Engineering*, vol. 51, no. 3, pp. 778–782, Mar. 2025, doi: https://doi.org/10.1109/tse.2025.3534032.

[7] Yulia Swandari, Ridi Ferdiana, and Adhistya Erna Permanasari, “Research Trends in Software Development Effort Estimation,” *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, vol. 10, pp. 625–630, Sep. 2023, doi: https://doi.org/10.1109/eecsi59885.2023.10295716.

[8] V. Berry, Arnaud Castelltort, B. Lange, J. Teriihoania, Chouki Tibermacine, and Catia Trubiani, “Is it Worth Migrating a Monolith to Microservices? An Experience Report on Performance, Availability and Energy Usage,” vol. 20, pp. 944–954, Jul. 2024, doi: https://doi.org/10.1109/icws62655.2024.00112.

[9] L. F. Fávero, N. Rodrigues, and F. J. Affonso, “A Systematic Mapping Study on the Modernization of Legacy Systems to Microservice Architecture,” *Applied System Innovation*, vol. 8, no. 4, pp. 86–86, Jun. 2025, doi: https://doi.org/10.3390/asi8040086.

[10] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” *Information and Software Technology*, vol. 137, p. 106600, Sep. 2021, doi: https://doi.org/10.1016/j.infsof.2021.106600.

[11] R. Aziz, V. Afzal, Z. A. Rana, and S. A. M. Gilani, “Improving Software Effort Estimation by Adjusting for Over-Commitment,” pp. 01–06, Dec. 2024, doi: https://doi.org/10.1109/fit63703.2024.10838423.

[12] Darina Bajusova, P. Silhavy, and R. Silhavy, “Enhancing Software Effort Estimation With Self-Organizing Migration Algorithm: A Comparative Analysis of COCOMO Models,” *IEEE Access*, vol. 12, pp. 67170–67188, Jan. 2024, doi: https://doi.org/10.1109/access.2024.3399060.

[13] Syed Sarmad Ali, J. Ren, K. Zhang, J. Wu, and C. Liu, “Heterogeneous Ensemble Model to Optimize Software Effort Estimation Accuracy,” *IEEE Access*, vol. 11, pp. 27759–27792, Jan. 2023, doi: https://doi.org/10.1109/access.2023.3256533.

[14] V. Raj and Hanumanthu Bhukya, “Assessing the Impact of Migration from SOA to Microservices Architecture,” *SN Computer Science*, vol. 4, no. 5, Jul. 2023, doi: https://doi.org/10.1007/s42979-023-01971-2.

[15] V. Velepucha and P. Flores, “A survey on microservices architecture: Principles, patterns and migration challenges.,” *IEEE Access*, vol. 11, pp. 1–1, 2023, doi: https://doi.org/10.1109/ACCESS.2023.3305687.

[16] Hamdy Michael Ayas, P. Leitner, and R. Hebig, “An empirical study of the systemic and technical migration towards microservices,” *Empirical Software Engineering*, vol. 28, no. 4, May 2023, doi: https://doi.org/10.1007/s10664-023-10308-9.

[17] V. Raj and K. Srinivasa Reddy, “Best Practices and Strategy for the Migration of Service-Oriented Architecture-Based Applications to Microservices Architecture,” *Algorithms for intelligent systems*, pp. 439–449, Jan. 2022, doi: https://doi.org/10.1007/978-981-16-7389-4_43.

[18] S. S. De Toledo, A. Martini, P. H. Nguyen, and D. I. K. Sjoberg, “Accumulation and Prioritization of Architectural Debt in Three Companies Migrating to Microservices,” *IEEE Access*, vol. 10, pp. 37422–37445, 2022, doi: https://doi.org/10.1109/access.2022.3158648.

[19] P. Di Francesco, P. Lago, and I. Malavolta, “Migrating Towards Microservice Architectures: An Industrial Survey,” *IEEE Xplore*, Apr. 01, 2018. https://ieeexplore.ieee.org/abstract/document/8417114/

[20] D. Faustino, N. Gonçalves, M. Portela, and A. Rito Silva, “Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation,” *Performance Evaluation*, vol. 164, p. 102411, May 2024, doi: https://doi.org/10.1016/j.peva.2024.102411.