Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 1, Issue 2

Open Access | PP: 97-103

DOI: https://doi.org/10.70315/uloap.ulirs.2024.0102012

A Methodology for Migrating Large Products to New Technology Stacks without Interrupting Development

Nizamutdinov Ilnar Rakipovich

Full Stack Developer, Belgrade, Serbia.

Abstract

The relevance of this research is determined by the growth of operational costs caused by the accumulation of technical debt in large monolithic software systems, which directly reduces competitiveness and slows down the time-to-market (TTM) of new products and functional changes. Unstable technical debt management practices, amplified by the fragmentation of business logic and the proliferation of isolated responsibility boundaries, transform into substantial direct and indirect financial losses. The aim of this study is to develop and empirically validate a comprehensive methodology for phased migration to modern technology stacks, ensuring continuity of development and operations (Zero Downtime) while simultaneously maintaining a high level of security and predictability of the production environment. The primary research approach is a systemic and comparative analysis of the architectural patterns Strangler Fig Pattern (SFP) and Branch by Abstraction (BbA), considered in conjunction with Data Driven Development practices, including the use of Feature Toggles and centralized server-side testing. Using the case of a large e-commerce service, the effects of implementing the proposed methodology are demonstrated: a 3–4-fold reduction in TTM (from 4–5 weeks to 1–1.5 weeks) and an approximately 4-fold decrease in labor effort required for implementing functional changes. It is shown that centralizing business logic and replacing 4,000 client-side tests with 1,000 server-side integration tests led to a 60% reduction in the number of critical incidents and to operational expenditure savings of more than 6 million USD annually. It is argued that the synergistic combination of the BbA and SFP patterns, complemented by the reengineering of the quality assurance system, constitutes a strategic and economically justified mechanism for the safe reduction of technical debt in high-load industrial systems. The results obtained have significant practical and theoretical value for chief architects, chief technology officers (CTOs), and researchers specializing in the modernization and evolution of complex software systems.

Keywords: *Non-Disruptive Migration, Technology Stack, Strangler Fig Pattern, Branch by Abstraction, Time to Market, Technical Debt, Microservice Architecture, Backend for Frontend, Automated Testing, Zero Downtime.*

INTRODUCTION

In the context of accelerating digital transformation and the dominance of Agile and DevOps methodologies, the ability of an organization to rapidly and controllably bring new features into production, thereby reducing Time to Market (TTM), acts as one of the key determinants of competitiveness [1]. At the same time, a significant proportion of large enterprises continues to rely on obsolete, monolithic, and highly coupled information systems that are characterized by a substantial volume of technical debt (TD). This debt manifests itself not only as a set of architectural and technological constraints, but also as a pronounced financial burden that provokes an exponential increase in operational expenditures and systematic inhibition of innovation activity [2]. Analytical reports of leading consulting companies state the necessity of consistently investing about 15% of the total IT budget in targeted and structured reduction of TD in order to maintain sustainable maturity of the digital core of the organization [3].

The impact of architectural constraints is particularly acute in large high load platforms, primarily in the e-commerce segment. In such systems, fragmentation of business logic and duplication of the same functionality across multiple client platforms (iOS, Android, Web) lead to a significant slowdown of TTM and complicate change management. In the examined case of the large e-commerce service Yandex.Market, the initial architecture required fourfold implementation of each new feature for different client applications, which increased TTM to 4–5 weeks and created a critical lag behind competitors that used centralized architectural solutions and were able to implement comparable changes within one week. An additional risk factor was architectural instability under high load conditions: according to internal estimates, one hour of platform downtime could lead to losses of about 130 million rubles in terms of GMV (Gross Merchandise Value). These circumstances demonstrate that systematic elimination of technical debt through architectural modernization is a strategic management priority that directly affects both market positions and financial sustainability of the company.

Analysis of the specialized literature shows that methods of incremental modernization aimed at reducing transformation risks have been developed in considerable detail. The most well known approach is the Strangler Fig Pattern proposed by Martin Fowler [4]. This pattern presupposes the phased replacement of the functionality of a monolithic system by new services using a facade or proxy layer for selective rerouting of calls, which makes it possible to gently decommission obsolete components [5]. This strategy is particularly effective when working with the external perimeter of the system and with consumer facing interfaces. At the same time, when it is necessary to modernize components that are deeply embedded in the code base and connected with a large number of upstream modules, it is advisable to apply the Branch by Abstraction pattern [7]. BbA enables deployment of a new implementation of a component that coexists with the old version for an extended period by introducing an abstraction layer, which allows switching between implementations without disrupting delivery continuity and without stopping the current release pipeline [8].

Despite the wide practical recognition of SFP and BbA, existing publications as a rule consider these patterns in isolation, without constructing a coherent integration model. There is a pronounced scientific and methodological gap associated with the absence of a systematized and empirically validated comprehensive methodology that, first, would combine SFP and BbA into a single architectural strategy for simultaneously solving the tasks of business logic unification and technology stack migration within one large scale project, and, second, would make it possible to quantitatively confirm the economic and operational effectiveness of such a synthesis in a real Data Driven Development (DDD) environment where continuity of A/B testing and of the release cycle is critically important. Insufficient elaboration of metrics, as well as the absence of established integration models for safe parallel coexistence of old and new functionality in high load DDD systems, forms a significant barrier for those who are responsible for architectural decision making.

Under these conditions, **the aim of the study** is formulated as the development and theoretical and empirical substantiation of a comprehensive methodology for migration of high load digital products to new technology stacks, with guaranteed continuity of development and operations, followed by verification of its effectiveness on the basis of a large industrial case. To achieve this aim, it is planned, first, to systematize the Strangler Fig and Branch by Abstraction architectural patterns that are relevant for non-disruptive modernization of components with different depth and density of dependencies; second, to develop a model for integrating A/B testing, Feature Toggles mechanisms, and progressive deployment (Canary Release) into the process of phased migration; third, to carry out a quantitative analysis of the impact of the developed methodology on key business indicators (TTM, total cost of changes, operational stability) using the example of a large e-commerce service; fourth, to substantiate the role of centralized server side integration testing as a critical element of the Zero Downtime methodology.

The scientific novelty of the study lies in the fact that, for the first time within a single methodological framework, the Branch by Abstraction and Strangler Fig architectural patterns are synthesized and empirically verified, integrated into the Data Driven Development paradigm, with demonstration of a fourfold reduction of Time to Market under conditions of large scale commercial operation.

The initial **author hypothesis** assumes that purposeful combined application of the Strangler Fig and Branch by Abstraction patterns, supported by centralized server side testing and a progressive deployment strategy, makes it possible to migrate mission critical systems without stopping the development process, while significantly reducing operational risks and labor costs of implementing changes.

MATERIALS AND METHODS

The study is based on a comprehensive methodological approach that integrates the principles of systems engineering, architectural modeling, and quantitative performance assessment. Comparative analysis was used as the primary toolset, allowing the characteristics of Time to Market, the volume of labor costs, and the level of operational risks before and after the introduction of modernization practices to be compared using the example of an e-commerce platform. Empirical data obtained from observing the dynamics of TTM, the frequency of critical incidents, and the structure of budget expenditures formed the factual basis for verifying the proposed methodology and assessing its effectiveness in a production environment.

The information base of the study was formed on the basis of a sample of 20 sources that met strict academic criteria of relevance (no more than 5 years old). Priority was given to publications in peer reviewed journals and high level digital libraries (in particular, IEEE, MDPI), as well as to works describing widely recognized architectural patterns and practices associated with Martin Fowler. For the quantitative substantiation of the economic significance of the issues under consideration and for confirming the validity of investments in the elimination of technical debt, analytical reports of leading consulting agencies were used, including materials from Accenture, which made it possible to correlate the results of the case analysis with the broader context of industry practice.

RESULTS AND DISCUSSION

The obtained results show that the integrated methodology based on the combination of the Strangler Fig (SFP) and Branch by Abstraction (BbA) patterns and the reengineering

of the quality system enabled large scale modernization of a high load e-commerce service without stopping the development pipeline, while a pronounced improvement in both economic and operational indicators was recorded. The central effect was the achievement of architectural unification and a significant reduction of Time to Market. The project on business logic unification was focused on eliminating code duplication across four client platforms (iOS, Android, Web Touch, Web Desktop). At the initial stage, each of these platforms functioned as a separate service with an autonomous implementation of business rules, which led to a fourfold increase in labor costs and to inconsistent product behavior: identical functions could interpret discounts, prices, or storefront display conditions in different ways.

To overcome these constraints, a Strangler Fig Pattern strategy was implemented through the introduction of a specialized architectural layer of the Backend for Frontend (BFF) type. The BFF facade ensured the migration of key, highly complex business logic to a centralized server side level, as a result of which client applications ceased to compute business rules independently and began to operate on unified data generated on the backend. This led to a radical reduction in the multiplicity of implementing functional changes. Whereas previously, for example, adding a new delivery block required about 20 person days of development and 12 person days of testing, that is, a total of about 32 person days across all platforms, after the transition to the centralized architecture an equivalent amount of work began to be performed in 8 person days, which corresponds to approximately a fourfold savings in resources. The aggregate effect of the implemented methodology, including its impact on operational and economic metrics, is summarized and presented in Table 1.

Table 1. Quantitative analysis of the impact of business logic unification on key development metrics (compiled by the author based on [1, 2])

Metric	Before unification (Fragmentation)	After unification (Centralization)	Effect	Related factor
Time to Market (TTM) for a new feature	4–5 weeks	1–1.5 weeks	Reduction by 3–4 times	Elimination of fourfold duplication of work
Labor costs per feature	32 person-days	8 person-days	Reduction by 4 times	Optimization of client teams' work (75% savings)
Annual payroll savings (client-side development)	N/A	~ $6,000,000	Reduction of excessive costs by 25–30%	Staff optimization (reduction from 200 to 80 developers)
Operational inconsistency	High	Minimized	Improvement of user experience	Standardization of functionality according to a catalog of reference behavior

The economic outcome of the transformation was driven not only by the reduction of Time to Market, but also by deep optimization of the organizational and staffing structure. The decrease in the volume of duplicative development led to a natural reduction in the need for distributed client teams: the number of frontend developers was reduced from 120 to 50, and the number of mobile development specialists from 80 to 30. The released human resources were purposefully reallocated towards strengthening the backend team, which, under conditions of centralized business logic, became the key limiting factor for scaling. The total annual effect in terms of reduced payroll expenses exceeded 6 million USD, which empirically confirms the proposition that architectural modernization functions not only as a technical, but also as a fundamental financial and strategic instrument of cost management.

Figure 1 demonstrates the impact of the introduction of unified business logic on Time to Market (TTM).

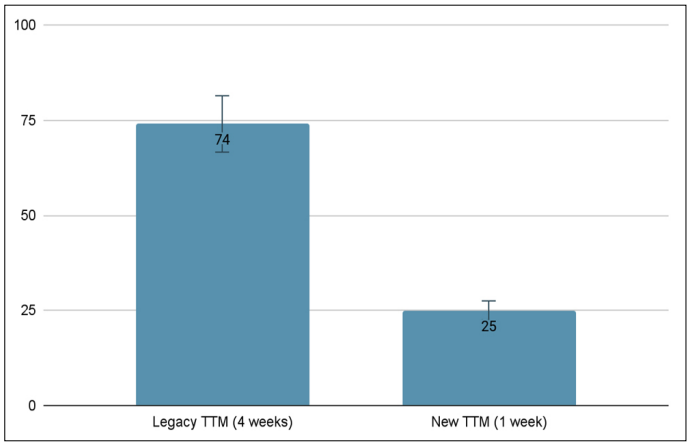

Fig. 1. Comparative analysis of Time to Market (TTM) before and after the implementation of unified business logic (author's data).

The next stage was an in-depth architectural modernization that included migrating the web platform from the legacy Yate framework to a modern technology stack (ReactJS

+ TypeScript). The high degree of Yate integration into the code base, the presence of thousands of components, and branching internal dependencies made the use of the Strangler Fig pattern practically inappropriate. Under these conditions, the Branch by Abstraction (BbA) methodology aimed at the safe replacement of deeply integrated modules was employed [7]. The application of BbA made it possible to deploy the new stack in parallel with the old one, maintaining the continuity of the release cycle and not disrupting the operation of the existing functionality.

The specificity of the Yandex.Market platform, which operates within the Data Driven Development paradigm and simultaneously supports dozens of A/B experiments, predetermined the need to ensure experiment-safe migration. Violation of the integrity of experimental configurations would have led to the necessity of restarting A/B tests and a significant delay of the project by weeks or months. To prevent such risks, the BbA methodology was integrated with the Feature Toggles mechanism, used in the role of Experiment Toggles [9]. The introduced Abstraction Layer performed the functions of an intelligent router: based on data about the user cohort determined by the active A/B experiment, requests were routed either to the original implementation on Yate or to the new implementation on ReactJS. This scheme provided the possibility of phased, incremental replacement of components, starting with less critical interface areas, while the correctness of the new stack was confirmed in the course of real A/B experiments. In addition, the option of targeted rollback to the old stack for selected user cohorts was preserved when defects or metric degradation were detected, which corresponds to the classical interpretation of BbA as a mechanism for controlled coexistence of the old and new implementations [7].

The results of applying the SFP and BbA architectural patterns formed a hybrid environment that significantly reduces modernization risks while simultaneously and radically increasing the requirements for the quality of integration testing [16]. The initial quality control model, based on a fragmented testing contour for four client platforms and comprising about 4000 disparate tests, was characterized by high maintenance costs and did not ensure guaranteed consistency of system behavior at the business logic level. Under such conditions, implementation of the Zero Downtime methodology required a targeted reengineering of the quality system with a fundamental shift of the testing focus to the server side. Instead of duplicating checks at the level of client applications, a set of 1000 centralized server side integration tests was developed, modeling full scale end-to-end (e2e) user scenarios from product search to order placement. These tests validate the correctness of the unified business logic, acting as the key mechanism for maintaining the integrity and predictability of system behavior under conditions of parallel coexistence of the old and new functionality.

Table 2 presents a comparison of testing efficiency using fragmented and centralized approaches.

Table 2. Comparison of testing efficiency: fragmented vs. centralized approach (compiled by the author based on [16–18])

Parameter	Fragmented testing (Legacy)	Centralized server-side testing (Unification)	Rationale
Testing point	Client platforms (4x)	Centralized Backend/API	Elimination of the need to duplicate functional checks
Test volume (coverage estimate)	~4000 (scattered, duplicative)	~1000 (highly effective integration tests)	Increased coverage density with lower maintenance volume
Costs for developing 1000 tests (estimate)	~$150,000 (for 4000 tests)	~$37,500 (for 1000 tests)	Savings on test development resources of more than $110,000
Reduction of critical incidents	Unstable	Reduction by >60%	Introduction of release-blocking e2e checks
Rollback requirements	Complex, lengthy	Fast rollback (Canary/ Feature Toggle)	Minimization of Time to Recovery (MTTR)

The financial effect of testing centralization manifested itself not only in increased predictability of the release cycle, but also in direct savings on the development of quality control tools. With an average developer productivity of four integration tests per day, constructing 4000 tests in the original fragmented architecture would have cost approximately 150,000 USD, whereas the development of 1000 new centralized server tests required about 37,500 USD. Thus, savings of more than 110,000 USD were achieved while maintaining, and in fact strengthening, the quality of coverage due to its concentration on the unified server side business logic.

The introduction of mandatory release blocking checks (mandatory checks) based on this set of centralized e2e tests made it possible to radically reduce operational risks. In the previous system configuration, critical errors in the Production environment occurred regularly due to inconsistencies in logic between different client platforms;

after the transition to the new model, the release pipeline became rigidly dependent on the successful completion of the full package of automated end-to-end tests, which technically blocks the deployment of incorrect changes. As a result, the frequency of critical incidents in the production environment decreased by more than 60%, indicating a substantial increase in the reliability of the platform.

The application of the non-disruptive migration methodology demonstrated that its success depends not only on technical solutions, but also on the level of organizational readiness [19]. In the case under analysis, the modernization outcome was closely linked to thoughtful human resource management: once business logic unification reduced the need for a large number of client developers, the released staff positions were purposefully reallocated to strengthen the backend team, whose size increased from 6 to 20 developers. This made it possible to eliminate the structural bottleneck arising from the concentration of all new centralized logic on the server side and to nearly triple the speed of backend development, thereby ensuring steady progress of the migration project.

The combination of the SFP and BbA patterns with progressive deployment practices further confirmed its effectiveness. Unlike more rigid and monolithic deployment strategies, the Canary Release approach makes it possible to controllably limit the impact area of potential defects by initially rolling out changes only to a small segment of the user base and gradually expanding coverage in the absence of negative effects [14]. This scheme significantly reduces risks when introducing both architectural changes and new functionality.

At the same time, a number of limitations were identified that must be taken into account when planning such modernization. First, managing a hybrid architecture during the phase of prolonged coexistence of old and new code, which can last for months or even years, inevitably increases the temporal and cognitive complexity of the code base and imposes higher requirements on the maintenance discipline of the Abstraction Layer responsible for routing and logic consistency [5]. Second, facade components — the abstraction layer in BbA and the BFF facade in SFP — form a single entry point into the system; with insufficient margin of robustness, redundancy, and fault tolerance, they themselves may transform into a new bottleneck or a single point of failure, which makes their careful design and operational monitoring critically important [5]. Third, adherence to Zero Downtime principles imposes strict constraints on data migration: all changes to schemas and data structures must be exclusively additive, excluding rapid destructive modifications. This requires detailed planning of migration steps, phased introduction of new fields and tables, and ensuring full backward compatibility between the old and new code over the entire horizon of the transition period [13].

The overall analysis of the obtained results shows that the methodology based on the synthesis of SFP and BbA, reinforced by centralization of the quality control system and flexible human resource management, ensures high speed, reliability, and economic efficiency of migration, allowing architectural modernization tasks to be solved without disrupting the continuity of development and operations.

CONCLUSION

The conducted study made it possible to formulate and empirically validate the effectiveness of a comprehensive methodology for non-disruptive migration of high-load software systems. The proposed approach is based on the strategic synthesis of the Strangler Fig Pattern (SFP) and Branch by Abstraction (BbA) architectural patterns, adapted to the conditions of continuous delivery and operation within the Data Driven Development (DDD) paradigm, which ensures alignment of architectural decisions with the requirements of the modern industrial development pipeline.

The results obtained confirm the initial hypothesis that the combined application of SFP and BbA in conjunction with centralized testing makes it possible to migrate mission-critical systems without stopping the development process. The SFP/BFF combination demonstrated its effectiveness in solving the problem of unifying and organizing the external perimeter of business logic, whereas BbA, integrated with Feature Toggles mechanisms, ensured the safe replacement of a deeply integrated technology stack, preserving the integrity of the experimental environment and enabling experiment-safe deployment.

From the standpoint of operational efficiency, the methodology led to a radical improvement in key business indicators. Time to Market for new features decreased by a factor of 3–4 (from 4–5 weeks to 1–1.5 weeks), and the labor costs for developing a single functional block were reduced from 32 to 8 person-days, that is, by a factor of four. Thus, it has been demonstrated that architectural transformation focused on centralizing business logic and controlled phased migration is directly converted into acceleration of the innovation cycle.

The economic component confirms that such modernization is not only technologically justified but also financially beneficial. Optimization of the staffing structure, made possible by eliminating multiple duplication of work on client platforms, in combination with the reengineering of the testing infrastructure, led to total annual savings in operating expenses exceeding 6 million USD. In this way, architectural modernization manifests itself as an element of long-term financial strategy rather than merely a local engineering project.

A critical factor in ensuring the Zero Downtime regime was the strategic shifting of the center of gravity of quality control to the server level. The replacement of 4000 disparate

client tests with 1000 centralized integration e2e checks with strictly defined SLAs made it possible to significantly reduce the probability of introducing inconsistent changes and to stabilize the release cycle. Empirically, a reduction of more than 60% in the number of critical incidents in the production environment was recorded, which indicates a substantial increase in the predictability and resilience of the system under load. Taken together, these results demonstrate that the comprehensive methodology of non-disruptive migration acts not merely as a set of engineering practices, but as a strategic instrument that enables large companies to systematically reduce technical debt, strengthen competitive positions, and maximize returns on investment in innovation.

Prospects for further research are associated with the development of formalized quantitative models that make it possible to determine the optimal moment for completing the coexistence period of the old and new architectures, minimizing overhead costs for maintaining a hybrid system. Additional scientific and practical interest is represented by the analysis of the potential of tools based on generative artificial intelligence (Generative AI) for automating the most labor-intensive stages of migration, such as constructing Abstraction Layers and automated generation of migration scripts. According to the forecasts of analytical agencies, such solutions are capable of significantly reducing the total costs of modernization and increasing the predictability of its outcomes.

REFERENCES

1. Wolfart, D., Assunção, W. K., da Silva, I. F., Domingos, D. C., Schmeing, E., Villaca, G. L. D., & Paza, D. D. N. (2021, June). Modernizing legacy systems with microservices: A roadmap. In Proceedings of the 25th international conference on evaluation and assessment in software engineering (pp. 149-159).https://doi.org/10.1145/3463274.3463334.
2. Allega, P., & Steinmetz, A. (n.d.). Enterprise architecture technical debt research. Gartner. https://www.gartner.com/en/information-technology/insights/technical-debt-reduction-ea-research (date accessed: September 09, 2024).
3. Accenture. (2024). Build your tech and balance your debt: Why balancing—not eliminating—tech debt is key to reinventing with a modern digital core. https://www.accenture.com/content/dam/accenture/final/accenture-com/document-3/Accenture-Build-Your-Tech-and-Manage-Your-Debt-2024.pdf (date accessed: September 10, 2024).
4. Fowler, M. (2024, August 22). Strangler Fig. martinfowler.com. https://martinfowler.com/bliki/StranglerFigApplication.html (date accessed: September 12, 2024).
5. Amazon Web Services. (n.d.). Strangler fig pattern. AWS Prescriptive Guidance. https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/strangler-fig.html (date accessed: September 14, 2024). docs.aws.amazon.com
6. Microsoft. (n.d.). Strangler fig pattern. Azure Architecture Center. https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig (date accessed: September 16, 2024).
7. Amazon Web Services. (n.d.). Branch by abstraction pattern. AWS Prescriptive Guidance. https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/branch-by-abstraction.html (date accessed: September 20, 2024).
8. Fowler, M. (2014, January 7). Branch by abstraction. martinfowler.com. https://martinfowler.com/bliki/BranchByAbstraction.html (date accessed: September 23, 2024).
9. Hodgson, P. (2017, October 9). Feature toggles (aka feature flags). martinfowler.com. https://martinfowler.com/articles/feature-toggles.html (date accessed: September 28, 2024).
10. Codefresh. (2023). Blue green deployment vs. canary: 5 key differences and how to choose. Codefresh Learning Center. https://codefresh.io/learn/software-deployment/blue-green-deployment-vs-canary-5-key-differences-and-how-to-choose/ (date accessed: October 04, 2024).
11. Alokai. (n.d.). Backend for frontend (BFF): What you need to know. Alokai Blog. https://alokai.com/blog/backend-for-frontend (date accessed: October 05, 2024).
12. ELITEX Systems. (n.d.). Backend for frontend (BFF)—Why is it essential to know? ELITEX Blog. https://elitex.systems/blog/backend-for-frontend-bff-everything-you-need-to-know (date accessed: October 20, 2024).
13. Naseer, U., Niccolini, L., Pant, U., Frindell, A., Dasineni, R., & Benson, T. A. (2020, July). Zero downtime release: Disruption-free load balancing of a multi-billion user website. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (pp. 529-541).
14. Octopus Deploy. (n.d.). Blue/green versus canary deployments: 6 differences and how to choose. Octopus Deploy. https://octopus.com/devops/software-deployments/blue-green-vs-canary-deployments/ (date accessed: October 20, 2024).
15. From monolithic systems to microservices: A comparative study of performance. (2020). Applied Sciences, 10(17), 5797. https://doi.org/10.3390/app10175797.

16. Software Engineering Institute. (2019). 8 steps for migrating existing applications to microservices. SEI Blog, Carnegie Mellon University. https://www.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/ (date accessed: October 22, 2024).

17. Daniel, B., Dig, D., Garcia, K., & Marinov, D. (2007). Automated testing of refactoring engines. In Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering (ESEC/FSE '07) (pp. 185–194). ACM. https://doi.org/10.1145/1287624.1287651.

18. Liu, K. (2021). Microservice migration patterns and how continuous integration and continuous delivery are affected: A case study of Indicio's journey towards microservice (Master's thesis). KTH Royal Institute of Technology. https://www.diva-portal.org/smash/get/diva2:1612613/FULLTEXT01.pdf (date accessed: October 24, 2024).

19. Ponce, F., Márquez, G., & Astudillo, H. (2019, November). Migrating from monolithic architecture to microservices: A Rapid Review. In 2019 38th International Conference of the Chilean Computer Science Society (SCCC) (pp. 1-7). IEEE. https://doi.org/10.1109/SCCC49216.2019.8966423.

20. Alonso, S., Kalinowski, M., Ferreira, B., Barbosa, S. D., & Lopes, H. (2023). A systematic mapping study and practitioner insights on the use of software engineering practices to develop MVPs. Information and Software Technology, 156, 107144.

21. Freire, A. F. A., Sampaio, A. F., Carvalho, L. H. L., Medeiros, O., & Mendonça, N. C. (2021). Migrating production monolithic systems to microservices using aspect oriented programming. Software: Practice and Experience, 51(6), 1280-1307. https://doi.org/10.1002/spe.2956.

22. Michael Ayas, H., Leitner, P., & Hebig, R. (2023). An empirical study of the systemic and technical migration towards microservices. Empirical Software Engineering, 28(4), 85.

Citation: Nizamutdinov Ilnar Rakipovich, "A Methodology for Migrating Large Products to New Technology Stacks without Interrupting Development", Universal Library of Innovative Research and Studies, 2024; 1(2): 97-103. DOI: https://doi.org/10.70315/uloap.ulirs.2024.0102012.