# Model-Driven Approaches for Serverless Software Development: Evaluation and Future Directions

Mehdi Eidi[a] and Raman Ramsin[b]
*Department of Computer Engineering, Sharif University of Technology, Azadi Avenue, Tehran, Iran*

Keywords: Model-Driven Development, Serverless Computing, Function-as-a-Service, Microservices Architecture.

Abstract: Serverless computing abstracts server management, enabling developers to focus on application logic. However, it introduces challenges across the software development lifecycle that necessitate the use of specialized methodologies. Model-Driven Development (MDD) is a promising approach, yet research on serverless-specific MDD remains underexplored, whereas methodologies for microservices are comparatively mature. This paper reviews selected MDD approaches for serverless and microservices development using a process-centered template and proposes an evaluation framework inspired by feature analysis, encompassing general, MDD, and serverless-related criteria. Applying the framework reveals gaps in existing serverless approaches and highlights adaptable strengths of microservices methodologies. These findings suggest directions toward mature MDD methodologies for serverless software development.

## 1 INTRODUCTION

Serverless computing is a cloud computing paradigm that abstracts server management from developers, enabling them to focus on application logic through Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS), which follow a pay-per-use model. Applications decompose into event-driven functions that scale elastically, often combined with traditional approaches in hybrid architectures (Casale et al., 2020; Hassan et al., 2021). Serverless introduces distinct engineering challenges requiring explicit guidance and automation (Wen et al., 2021); it is a shift in methodology, not just in infrastructure.

Model-Driven Development (MDD) addresses complexity by treating models as primary artifacts and automating transformations and code generation. Model-Driven Architecture (MDA) organizes development across *computation-independent* (CIM), *platform-independent* (PIM), and *platform-specific models* (PSM), with transformations connecting requirements, design, and implementation (Asadi & Ramsin, 2008; Selic, 2003). MDD is a natural candidate for addressing serverless challenges, but research is scarce. Microservices and serverless share common foundations and are often combined in hybrid architectures, but serverless introduces unique challenges not fully addressed by microservices.

This paper reviews selected MDD approaches for serverless and microservices development using a process-centered template (Ramsin & Paige, 2008). We also propose an evaluation framework for assessing approaches against explicitly defined criteria, iteratively refined across general software development, MDD-related, and serverless-related concerns (Linkman et al., 1997). Our analysis reveals that while many approaches support design-time modeling and deployment automation, they are weak in other lifecycle activities. The paper contributes a structured review, evaluation framework, and comparative assessment identifying key gaps and future directions toward more mature MDD methodologies for serverless software development.

## 2 REVIEW OF APPROACHES

This section reviews MDD approaches for serverless software development using a process-centered template. Since dedicated serverless MDD research is scarce, we also examine microservice-based MDD methodologies sharing architectural foundations with serverless; this helps reveal methodological gaps and leverage mature techniques to advance MDD

[a] https://orcid.org/0009-0000-2426-7728
[b] https://orcid.org/0000-0003-1996-9906

Eidi, M. and Ramsin, R.
Model-Driven Approaches for Serverless Software Development: Evaluation and Future Directions.
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 14th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2026)*, pages 560-567
ISBN: 978-989-758-798-6; ISSN: 2184-4348
Proceedings Copyright © 2026 by SCITEPRESS – Science and Technology Publications, Lda.

approaches for serverless software development.

## 2.1 RADON Methodology

RADON provides a process for engineering microservices and serverless applications (Casale et al., 2020). Its workflows include (Figure 1): *Application Development*: models serverless and microservices architectures using TOSCA. *Verification*: checks models against constraints. *Decomposition*: supports breaking monoliths into microservices/functions and runtime-based refinement. *Defect Prediction*: uses machine learning to identify failure-prone configuration scripts. *Continuous Testing*: defines and executes automated tests for services, functions, and pipelines. *Monitoring*: gathers runtime metrics, generating dashboards and alerts for quality assurance. *CI/CD*: integrates workflows into automated pipelines for testing, verification, packaging, and deployment.

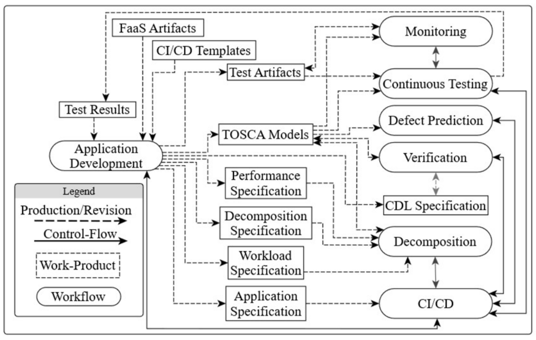

Figure 1: RADON, adapted from (RADON, 2021).

## 2.2 Standards-Based Method

Yussupov et al. introduce a method for developing serverless workflows using BPMN and TOSCA (Yussupov et al., 2022). The method includes two phases (Figure 2). The *Workflow Modeling* phase includes: *BPMN Modeling*: creates a generic BPMN function orchestration model; *Orchestrator-Specific Generation*: produces provider-specific models. The *Deployment Modeling* phase includes: *TOSCA Deployment Modeling*: specifies deployment as a service template with function code and orchestration models; *Packaging*: exports the TOSCA model as a CSAR deployment package; *Enactment*: deploys the application and executes the orchestration.

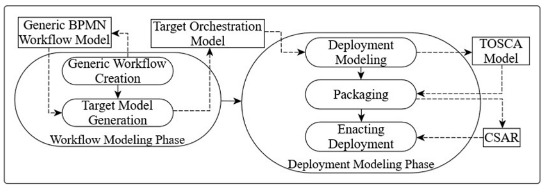

Figure 2: Standards-based modeling and deployment, adapted from (Yussupov et al., 2022).

## 2.3 UMLPDA Framework

UMLPDA is a framework for developing data-driven applications in serverless environments (Samea et al., 2020). It introduces a UML profile for the PIM and generates applications using a transformation engine in four stages (Figure 3): *Modeling*: captures requirements and creates PIMs. *Define Rules*: describes translation of stereotypes to low-level artifacts. *Transformation*: generates code. *Deploy*: deploys the application on a serverless platform.

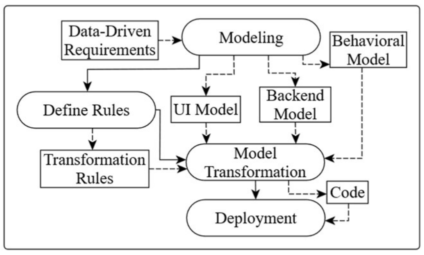

Figure 3: UMLPDA, adapted from (Samea et al., 2020).

## 2.4 UMLPMSC Profile

UMLPMSC targets the service-configuration layer of event-driven serverless applications deployed across multiple clouds, addressing low-level complexity by raising abstraction to UML models transformable into serverless configurations (Samea et al., 2019). Service configuration requirements are captured using a dedicated UML profile. UMLPMSC reduces configuration complexity and improves portability by enabling automated PIM-to-PSM transformations.

## 2.5 Serverless4IoT Approach

Serverless4IoT supports specification, deployment, and maintenance of hybrid applications combining serverless functions with traditional components across the cloud-edge-IoT continuum (Ferry et al., 2022). It introduces *deployment templates* supporting reusable, customizable software stacks for serverless functions. Developers model deployments in the IDE; the orchestrator distributes artifacts; gateways run local services; function deployment is automated via GeneSIS; and a models@run.time mechanism keeps models synchronized for orchestration (Figure 4).

## 2.6 DevOps-Enabled Methodology

Rademacher et al.'s process derives microservice

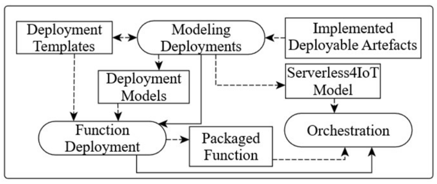

Figure 4: Serverless4IoT, adapted from (Ferry et al., 2022).

code from underspecified domain models (Rademacher et al., 2020). It provides modeling languages (LEMMA) and automated transformations across four phases (Figure 5): *Domain Modeling*: creates a domain-driven design (DDD) domain model using a UML profile; model transformation derives a LEMMA domain model from each bounded context. *DevOps Model Derivation*: generates service and operation models aligned with DevOps viewpoints. *Model Adaptation*: refines derived models to resolve underspecification. *Code Generation*: attaches implementation details via technology and mapping models; transformations generate microservice code.

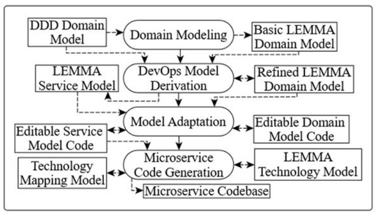

Figure 5: DevOps-enabled methodology using LEMMA, adapted from (Rademacher et al., 2020).

## 2.7 GenMicro Model-Driven Approach

GenMicro provides microservice architecture (MSA) generation (Kungne et al., 2025). Following MDA, it defines PIMs, transforms them into PSMs, and refines them into Java code across three phases (Figure 6): *PIM Formalization*: models MSAs using class diagrams and specific microservice node types. *PSM Modeling*: refines PIM elements into object-oriented classes across layers. *Refinement*: introduces platform details and generates deployable artifacts.

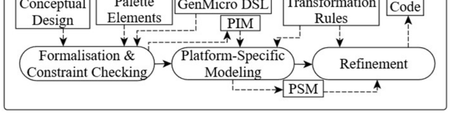

Figure 6: GenMicro, adapted from (Kungne et al., 2025).

## 2.8 Zynerator Model-Driven Approach

Zynerator extends MDA to microservices (Zouani & Lachgar, 2024). It relies on *semantic decorators* for enriching the PIM with functional and non-functional semantics, and *templates* for projecting the refined PIM onto target technologies across three phases (Figure 7): *Metadata Definition*: establishes initial configuration. *Design and Decoration*: introduces a PIM with semantic tags like CACHEABLE, guiding requirement interpretation. *Target Technology Projection*: maps the analyzed model to templates, producing frontend/backend/deployment artifacts.

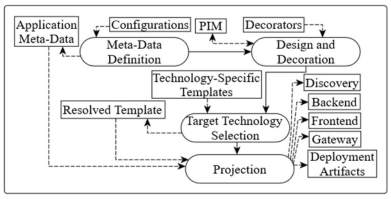

Figure 7: Zynerator, adapted from (Zouani & Lachgar, 2024).

## 3 EVALUATION

We adopted an evaluation method inspired by *Feature Analysis* (Linkman et al., 1997) to assess the approaches reviewed in the previous section. Feature Analysis evaluates methods and tools through two forms: simple form using binary judgements, and scale form using numerical scales to express conformance to features. The reviewed MDD approaches were evaluated against Evaluation Criteria (EC) of three types: (1) *simple:* yes/no assessment; (2) *scale*: degree of presence on three levels: (A) full coverage, (B) partial coverage, (C) no coverage; and (3) *narrative*: descriptive analysis.

Three categories of criteria have been used for evaluation: general, MDD-related, and serverless-related. Criteria for *general* and *model-driven software development* were adopted from (Asadi & Ramsin, 2008; Ramsin & Paige, 2010), while additional *serverless-specific* criteria were collected by studying surveys and existing approaches. Relevant resources were iteratively examined to identify additional criteria and refine existing ones. Tables 1, 2, and 3 present the final evaluation criteria.

Our evaluation criteria adhere to the four validity meta-criteria of (Karam & Casselman, 1993), in that they are: (1) sufficiently *general* to apply to all MDD methodologies; (2) *precise* enough to distinguish similarities and differences; (3) *comprehensive* in covering significant features; and (4) *balanced*, ensuring appropriate attention to technical, managerial, and usage aspects of methodologies.

Tables 4, 5, and 6 summarize the evaluation results. Interpretation is context-dependent and can support requirements-based methodology selection or identification of limitations and gaps. The evaluation framework can also guide *Method Engineering* efforts for adapting, extending, meta-modeling, and assembling model-driven processes for serverless software development (Asadi & Ramsin, 2008).

Table 1: General software development evaluation criteria.

| Criterion Name | | | Type | Criterion Definition |
|---|---|---|---|---|
| Coverage | Generic Life Cycle | Requirements Engineering | Scale | Ability to support collecting/documenting stakeholders' needs/expectations |
| | | Analysis | Scale | Ability to analyze and refine requirements to produce precise descriptions |
| | | Design | Scale | Ability to transform requirements into an architecture and system design |
| | | Implementation | Scale | Ability to facilitate implementing the system based on the design |
| | | Test | Scale | Support for creating/executing tests to ensure the system meets requirements |
| | | Deployment | Scale | Ability to automate and manage software deployment across environments |
| | | Maintenance | Scale | Ability to track, manage, and resolve issues or changes after deployment |
| | Umbrella Activitiess | Project Management | Scale | Features that enable planning, organizing, and controlling activities |
| | | Quality Assurance | Scale | Ensuring that the development process adheres to defined standards/practices |
| | | Risk Management | Scale | Support for identifying/assessing/managing potential risks during the process |
| Reusability | | | Scale | Support for creating and documenting reusable artifacts for future projects |
| User Involvement Support | | | Simple | Encouragement of active user involvement. |
| Adaptability | | | Scale | Methodology's ability to be configured or tailored to project-specific needs |
| Completeness of Definition | | | Scale | Degree to which the process is documented with clear, logical, and detailed descriptions |
| Definition Type | | | Narrative | (A) Process-oriented or (B) Product-oriented |
| Traceability | | | Scale | How traceable the artifacts are to the requirements |
| Based on Requirements | | | Scale | Early identification and separate modeling of requirements as the basis for development |
| Complexity Management | | | Scale | Keeping work-unit complexity manageable through decomposition/layering |
| Methodology Type | | | Scale | (A) Pure model-driven method or (B) Extension of traditional methods |
| Application Scope | | | Narrative | (A) Domain-specific or (B) General-purpose |

Table 2: MDD-related evaluation criteria.

| Criterion Name | | Type | Criterion Definition |
|---|---|---|---|
| Modeling Layers | Definition of Modeling Boundaries | Simple | Ability to define clear and effective modeling boundaries |
| | CIM Creation | Scale | Support for creating high-level, computation-independent models |
| | PIM Creation | Scale | Support for creating PIM models |
| | PSM Creation | Scale | Support for creating PSM models that describe implementation details |
| | Code Generation | Scale | Ability to generate executable code based on created models |
| Transformations Definition | CIM to PIM | Scale | Ability to define and perform transformations from CIM to PIM |
| | PIM to PSM | Scale | Ability to define and perform transformations from PIM to PSM |
| | PSM to Code | Scale | Ability to define and perform transformations from PSM to code |
| Round-Trip Engineering | | Scale | Support for bidirectional transformations ensuring consistency |
| Source-Target Model Synchronization | | Scale | Ability to synchronize changes effectively between source and target models |
| Degree of Code Automation | | Scale | Extent to which final code generation is automated |
| Models Quality | Structural, Functional, Behavioral | Scale | Comprehensive coverage of different modeling dimensions |
| | Logical | Scale | Degree of model abstraction and logical clarity |
| Cognitive Load | Documentation & Guidelines | Scale | Completeness of educational material required to understand the process |
| | Process Familiarity | Scale | How familiar or intuitive the process is, influencing learning speed |
| | Informational Feedback | Scale | Provision of timely and sufficiently informative feedback by the tool |
| Coverage of Metamodels in Domain | | Scale | Extent to which the metamodels cover the problem domain |
| Tool | All-Levels Transformations | Scale | Ability of the tool to perform transformations across all abstraction levels |
| | Model Evaluation | Scale | Capability to assess and analyze model quality and consistency |
| | Metadata Management | Scale | Ability to manage metadata associated with models |
| | Automated Testing | Scale | Support for automated testing of models and transformations |
| | Traceability | Scale | Ability to establish/maintain traceability links between models at different levels |
| Standard Definitions Compliance | | Scale | Adherence to standard definitions in MDD |
| Model-Based Testing | | Scale | Support for generating and executing tests derived from models |

Table 3: Serverless-related evaluation criteria.

| Criterion Name | | | Type | Criterion Definition |
|---|---|---|---|---|
| Exploration | Serverless Suitability Analysis | | Scale | Extent to which the method enables deciding on serverless suitability |
| | Serverless Cost Analysis | | Scale | Ability to estimate serverless costs from expected usage |
| | Serverless Risk Analysis | | Scale | Degree to which risks of serverless development are identified |
| | Cloud Provider Selection | | Scale | Ability to compare/select providers based on constraints/offerings |
| Requirements | Event-Driven Requirements Capture | | Scale | Ability to capture requirements centred on business events |
| | Fine-Grained NFR Capture | | Scale | Ability to link detailed and fine-grained NFRs to architectural choices |
| Modeling | Event Modeling | | Scale | Ability to model events, triggers, and event sources |
| | Function Granularity Determination | | Scale | Offering patterns and guidelines to determine appropriate function granularity |
| | Domain-Driven Design | | Scale | Supporting domain modeling and domain-driven design. |
| | Workflow Modeling | | Scale | Ability to model workflows combining functions into complex execution logic |
| | Choreography Modeling | | Scale | Ability to model decentralized event-based choreography of functions |
| | Configuration Modeling | | Scale | Ability to model unified configurations of large numbers of serverless functions |
| | Data Pipeline Modeling | | Scale | Ability to model data pipelines/streams/queues, and serverless dataflows |
| | State Management Modeling | | Scale | Ability to model patterns and strategies for managing serverless state |
| | Event Contracts | | Scale | Capability to define event contracts and message schema specifications |
| | Multi-Cloud Modeling | | Scale | Ability to model multi-cloud architectures across serverless providers |
| | Fault Modeling | | Scale | Ability to model retry policies, backoff, dead-letter queues, and compensation |
| Design | External Service Integration | | Scale | Ability to model interactions with external systems in hybrid architectures |
| | Serverless Patterns Support | | Scale | Capability to design using established serverless architectural patterns |
| | Reusable Function Design | | Scale | Ability to design reusable functions improving shared organizational capabilities |
| Security | Fine-Grained Access Configuration | | Scale | Capability to enforce fine-grained least-privilege access for functions/services |
| | Secrets Management | | Scale | Ability to manage secrets and sensitive data injection into serverless functions |
| Implementation | Workflow Generation | | Scale | Ability to automatically generate low-level workflow implementation code |
| | Deployment Artifact Generation | | Scale | Ability to generate automated IaC deployment artifacts for serverless setups |
| Test | Workflow Testing | | Scale | Capability to test workflows and validate their execution correctness |
| | Serverless-Specific Testing | | Scale | Ability to execute serverless-specific tests addressing cold start and latency |
| Deployment | CI/CD Support | | Scale | Ability to define CI/CD pipelines and automate deployment processes |
| | Deployment Strategies | | Scale | Capability to apply deployment strategies such as canary or blue-green releases |
| | Function Versioning | | Scale | Ability to manage function versioning and operational aliases |
| Maintenance | Observability | Distributed Tracing | Scale | Support for distributed tracing, KPI monitoring, and event logging |
| | | Logging | | |
| | Feedback Loop | | Scale | Support for using runtime feedback to refine architecture |
| Cold-Start Mitigation | | | Scale | Ability to mitigate cold-start impact for latency-sensitive serverless functions |
| Vendor Lock-In Mitigation | | | Scale | Ability to reduce vendor lock-in and enhance solution portability |

Table 4: Results of applying the general evaluation criteria.

| Criterion | | Methodology | RADON | Standards-Based | UMLPDA | UMLPMSC | Serverless4IoT | DevOps-Enabled | GenMicro | Zynerator |
|---|---|---|---|---|---|---|---|---|---|---|
| Coverage | Generic Life Cycle | Requirements Eng. | B | C | C | C | C | B | C | B |
| | | Analysis | C | C | C | C | C | B | C | C |
| | | Design | A | B | A | B | B | A | A | A |
| | | Implementation | A | B | A | C | C | A | A | A |
| | | Test | A | C | C | C | C | C | C | B |
| | | Deployment | A | A | C | B | A | A | B | A |
| | | Maintenance | B | C | C | C | B | B | C | C |
| | Umbrella Activities | Project Management | C | C | C | C | C | C | C | C |
| | | Quality Assurance | A | B | C | C | C | B | B | B |
| | | Risk Management | C | C | C | C | C | C | C | C |
| Reusability | | | A | A | B | B | A | A | C | B |
| User Involvement Support | | | No | No | No | No | No | Yes | No | No |
| Adaptability | | | A | A | C | B | C | B | C | B |
| Completeness of Definition | | | A | A | B | B | B | A | B | B |
| Definition Type | | | A | A | A | B | B | A | A | A |
| Traceability | | | C | C | C | C | C | B | B | B |
| Based on Requirements | | | B | C | B | C | C | B | C | B |
| Complexity Management | | | B | A | B | B | B | A | B | B |
| Methodology Type | | | A | A | A | A | A | A | A | A |
| Application Scope | | | A | A | A | A | A | A | A | A |

Table 5: Results of applying the MDD-related evaluation criteria.

| Criterion | Methodology | RADON | Standards-Based | UMLPDA | UMLPMSC | Serverless4IoT | DevOps-Enabled | GenMicro | Zynerator |
|---|---|---|---|---|---|---|---|---|---|
| Modeling Layers | Def. of Model Boundaries | No | No | No | No | No | Yes | Yes | Yes |
| | CIM Creation | C | C | C | C | C | A | C | C |
| | PIM Creation | B | A | A | A | B | A | A | A |
| | PSM Creation | A | B | B | C | B | A | A | B |
| | Code Generation | B | A | A | C | C | A | A | A |
| Transformations Definition | CIM to PIM | C | C | C | C | C | B | C | C |
| | PIM to PSM | B | B | C | C | C | B | A | B |
| | PSM to Code | B | B | B | C | C | A | B | B |
| Round-Trip Engineering | | C | C | C | C | A | C | C | C |
| Source-Target Model Synch. | | C | C | C | C | A | B | B | C |
| Degree of Code Automation | | B | B | B | C | C | B | B | A |
| Models Quality | Structural, Functional, Behavioral | B | B | A | B | B | A | A | B |
| | Logical | B | B | B | B | B | A | B | B |
| Cognitive Load | Documentation&Guidelines | A | A | B | B | B | A | B | B |
| | Process Familiarity | C | B | A | A | C | B | B | C |
| | Informational Feedback | A | A | C | C | C | B | C | C |
| Coverage of Metamodels | | B | B | B | B | B | A | B | B |
| Tool | All-Levels Transformations | C | B | B | C | C | B | B | B |
| | Model Evaluation | A | C | C | C | C | B | C | C |
| | Metadata Management | B | C | C | C | C | B | C | C |
| | Automated Testing | A | C | C | C | C | C | C | C |
| | Traceability | C | C | C | C | A | B | C | C |
| Standard Definitions Compliance | | C | B | B | A | C | B | A | B |
| Model-Based Testing | | A | C | C | C | C | C | C | C |

Table 6: Results of applying the serverless-related evaluation criteria.

| Methodology / Criterion | | | RADON | Standards -Based | UMLPDA | UMLPMSC | Serverless4IoT | DevOps-Enabled | GenMicro | Zynerator |
|---|---|---|---|---|---|---|---|---|---|---|
| Exploration | Serverless Suitability Analysis | | C | C | C | C | C | C | C | C |
| | Serverless Cost Analysis | | C | C | C | C | C | C | C | C |
| | Serverless Risk Analysis | | C | C | C | C | C | C | C | C |
| | Cloud Provider Selection | | C | C | C | C | C | C | C | C |
| Requirements | Event-Driven Capture | | C | C | C | C | C | C | C | C |
| | Fine-Grained NFR Capture | | A | C | C | B | B | C | C | B |
| Modeling | Event Modeling | | A | A | B | A | B | B | C | C |
| | Granularity Determination | | A | C | C | C | C | B | C | C |
| | Domain-Driven Design | | C | C | C | C | C | A | B | C |
| | Workflow Modeling | | A | A | B | B | C | C | B | C |
| | Choreography Modeling | | C | C | C | B | C | C | C | C |
| | Configuration Modeling | | A | A | C | A | B | B | B | B |
| | Data Pipeline Modeling | | A | A | C | A | C | C | C | C |
| | State Management Modeling | | C | C | B | B | C | C | C | C |
| | Event Contracts | | C | B | C | C | C | C | C | C |
| | Multi-Cloud Modeling | | A | A | C | A | B | C | C | C |
| | Fault Modeling | | C | A | B | C | C | C | C | C |
| Design | External Service Integration | | A | A | A | A | A | A | B | B |
| | Serverless Patterns Support | | B | A | C | C | C | C | C | C |
| | Reusable Function Design | | B | B | C | C | B | C | C | C |
| Security | Fine-Grained Access | | B | C | C | A | C | C | C | B |
| | Secrets Management | | A | C | C | B | C | C | C | C |
| Implementation | Workflow Generation | | B | A | C | C | C | C | B | C |
| | Deployment Artifact Generation | | A | A | C | C | A | B | B | B |
| Test | Workflow Testing | | A | C | C | C | C | C | C | C |
| | Serverless-Specific Testing | | B | C | C | C | C | C | C | C |
| Deployment | CI/CD Support | | A | C | C | C | C | C | C | B |
| | Deployment Strategies | | C | C | C | C | C | C | C | C |
| | Function Versioning | | A | C | C | C | C | C | C | C |
| Maintenance | Observability | Tracing | C | C | C | C | B | C | C | C |
| | | Logging | A | C | C | B | C | C | C | B |
| | Feedback Loop | | A | C | C | C | B | C | C | C |
| Cold-Start Mitigation | | | C | C | C | C | C | C | C | C |
| Vendor Lock-In Mitigation | | | A | A | C | B | A | B | C | B |

## 4 CONCLUSION

This paper reviewed MDD approaches for serverless and microservice development using a process-centered template and introduced a criteria-based evaluation framework. The overall evaluation results indicate that there is currently no mature methodology for serverless software development that performs well across all three subsets of the evaluation criteria. At the same time, the results clarify the strengths of existing methods, such as the greater MDD-related maturity observed in microservices approaches, while also highlighting the shortcomings that need to be addressed.

Overall, RADON provides the broadest support when general and serverless-related perspectives are considered together, while the DevOps-Enabled methodology stands out most clearly on the MDD-related criteria, particularly those concerning modeling structure and disciplined use of modeling layers. The Standards-Based approach also performs consistently well, especially on serverless workflow modeling and deployment concerns. In contrast, others tend to exhibit more specialized strengths but leave substantial portions of the criteria only partially addressed or entirely unaddressed.

As future work, based on the findings of this analysis and by leveraging the strengths of existing approaches, we intend to propose an MDD methodology for serverless software development that addresses the identified shortcomings.

## REFERENCES

Asadi, M., & Ramsin, R. (2008). MDA-Based Methodologies: An Analytical Survey. In *Model Driven Architecture – Foundations and Applications* (Vol. 5095, pp. 419–431). Springer.

Casale, G., Artač, M., van den Heuvel, W.-J., van Hoorn, A., Jakovits, P., Leymann, F., Long, M., Papanikolaou, V., Presenza, D., Russo, A., Srirama, S. N., Tamburri, D. A., Wurster, M., & Zhu, L. (2020). RADON: Rational decomposition and orchestration for serverless computing. *SICS Software-Intensive Cyber-Physical Systems*, 35, 77–87.

Ferry, N., Dautov, R., & Song, H. (2022). Towards a Model-Based Serverless Platform for the Cloud-Edge-IoT Continuum. In *22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 851–858.

Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. *Journal of Cloud Computing*, 10(1), Article 39.

Karam, G. M., & Casselman, R. S. (1993). A cataloging framework for software development methods. *IEEE Computer*, 26(2), 34–44.

Kungne, W. K., Kouamou, G.-E., & Ayang, P. (2025). GenMicro: A Tool for Generating Microservice Architectures with In-Depth Microservice Design. *Journal of Computer Science*, 21(3), 729–740.

Linkman, S., Kitchenham, B., & Law, D. (1997). DESMET: A methodology for evaluating software engineering methods and tools. *Computing & Control Engineering Journal*, 8(3), 120–126.

Rademacher, F., Sachweh, S., & Zundorf, A. (2020). Deriving Microservice Code from Underspecified Domain Models Using DevOps-Enabled Modeling Languages and Model Transformations. In *46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 229–236.

RADON. (2021). Deliverable D3.1: RADON methodology [Public Deliverable]. *RADON Project (Horizon 2020 Grant Agreement No. 825040).* https://radon-h2020.eu/public-deliverables/

Ramsin, R., & Paige, R. F. (2008). Process-centered review of object oriented software development methodologies. *ACM Computing Surveys*, 40(1), 1–89.

Ramsin, R., & Paige, R. F. (2010). Iterative criteria-based approach to engineering the requirements of software development methodologies. *IET Software*, 4(2), 91–104.

Samea, F., Azam, F., Anwar, M. W., Khan, M., & Rashid, M. (2019). A UML Profile for Multi-Cloud Service Configuration (UMLPMSC) in Event-driven Serverless Applications. In *8th International Conference on Software and Computer Applications*, 431–435.

Samea, F., Azam, F., Rashid, M., Anwar, M. W., Haider Butt, W., & Muzaffar, A. W. (2020). A model-driven framework for data-driven applications in serverless cloud computing. *PLOS ONE*, 15(8), e0237317.

Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5), 19–25.

Wen, J., Chen, Z., Liu, Y., Lou, Y., Ma, Y., Huang, G., Jin, X., & Liu, X. (2021). An empirical study on challenges of application development in serverless computing. In *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 416–428.

Yussupov, V., Soldani, J., Breitenbücher, U., & Leymann, F. (2022). Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA. *Software: Practice and Experience*, 52(6), 1454–1495.

Zouani, Y., & Lachgar, M. (2024). Zynerator: Bridging Model-Driven Architecture and Microservices for Enhanced Software Development. *Electronics*, 13(12), 2237.