Information technologies in economics and environmental sciences

Journal homepage: https://journals.nubip.edu.ua/index.php/Inf/
ISSN 3083-7502

COMPUTER SCIENCE ◈ КОМП'ЮТЕРНІ НАУКИ

Received 2026-02-09 Accepted 2026-03-16 Published 2026-04-22

UDC 004.4:004.9 DOI: 10.31548/itees.2026.01.009

Nazarenko Volodymyr
Ph.D., Associate Professor of the Department of Computer Systems, Networks and Cybersecurity Department,
National University of Life and Environmental Sciences of Ukraine
ORCID: https://orcid.org/0000-0002-7433-2484
E-mail: volodnz@nubip.edu.ua

A UE5 DEVELOPMENT ADOPTION FRAMEWORK FOR MULTIPLAYER DEVELOPMENT: REPLICATION, SERVICES, OBSERVABILITY, AND LIVEOPS

Abstract. *This paper proposes a practical, UE5-centered adoption framework that translates modern software development methodologies from Web/App, enterprise/heavy systems, AI systems, and contemporary game development into measurable production practices for multiplayer Unreal Engine 5 (UE5) teams. While Agile, MVP, DevOps, microservices, data-driven development, and architectural patterns are widely discussed, UE5 multiplayer imposes hard constraints that are often absent from generic methodology guidance: replication cost and relevance/dormancy management, server-authoritative state, dedicated server build and deployment operations, online session flows, and continuous delivery/LiveOps expectations. To address this "methodology translation" gap, we define a software-focused research design that operationalizes methodology adoption through implementable UE5 artifacts: a suite of test projects, multiplayer archetypes, and an instrumentation plugin stack for deterministic scenario execution, controlled network stress injection, replication observability, and standardized telemetry outputs. We present pilot-format result templates and case-study tables that connect methodology choices to objective UE5 indicators, including replication budget per client, temporal consistency of server simulation steps (server tick rate stability), session join reliability, build-to-playable time, crash-free server hours, and change lead time. A Lyra-style GameFeature plugin slice is introduced to demonstrate clean architectural boundaries and auditable iteration throughput via feature lifecycle telemetry tied to CI build identifiers. The paper synthesizes these results into a staged adoption map that prioritizes replication-first KISS discipline, multiplayer MVP defined as an operations-capable vertical slice, minimal services before microservice decomposition, data-driven tuning with safe rollbacks, and modular feature slices for scalable iteration. This framework is intended as a practical guide for UE5 multiplayer teams and as a foundation for larger empirical evaluations.*

Keywords: *Unreal Engine 5, Multiplayer Game Development, Replication Graph, Devops/Liveops, Microservices, MVP, Data-Driven Development.*

Introduction. Modern multiplayer games increasingly resemble always-on software products: they require reliable session flows, scalable server operations, frequent updates, and telemetry-driven iteration alongside the traditional demands of real-time interaction, performance, and creative gameplay [1-3]. Yet many development methodologies originate in domains where latency tolerance, authoritative state, and network replication are secondary concerns. Unreal Engine 5 (UE5) provides robust networking capabilities. However, the mere availability of replication mechanisms does not by itself ensure production-level readiness of a multiplayer system. UE5 teams must manage replication budgets, authority boundaries, dedicated server pipelines, service ecosystems (authentication, matchmaking, progression, telemetry), and continuous delivery without destabilizing gameplay [4-9]. Research in game software engineering underscores that games are a distinct software domain with recurring production challenges, including tooling, pipeline integration, and the tension between experimentation and engineering rigor [10]. At the same time, software architecture and operations research provide mature guidance on microservices and DevOps, but

typically without UE5's replication and dedicated server constraints [11]. This creates a practical gap: teams know what methodologies exist but lack a UE5-specific translation that turns them into implementable engineering decisions and measurable outcomes.

Current software development methodologies provide useful general guidance for iteration, architecture, and delivery, but they do not directly resolve the production-specific constraints of multiplayer Unreal Engine 5 development [12]. In UE5 multiplayer projects, engineering decisions are shaped not only by feature scope but also by replication cost, server-authoritative state management, dedicated server deployment, session reliability, telemetry coverage, and LiveOps continuity. The research problem addressed in this paper is, therefore, the lack of a UE5-oriented methodological framework that connects general development principles with measurable multiplayer engineering outcomes.

Purpose. The purpose of this research is to systematically synthesize and operationalize modern software development methodologies into a practical adoption framework for multiplayer UE5 production, where replication, dedicated server operations, service boundaries, and continuous delivery/LiveOps are first-class constraints.

Literature review. Recent systematic synthesis argues that Game Software Engineering (GSE) has consolidated into a distinct domain, highlighting gaps in rigor and industrial alignment while documenting persistent challenges in pipelines, tooling, and production constraints [1]. Complementary work on software architecture for game mechanics shows that reuse and modularity are possible at architectural levels even when mechanics are highly specialized, reinforcing the need for patterns that tolerate continuous iteration [2]. Research on architecting microservices emphasizes that decomposition, coupling, and adoption are complex in practice; benefits come with governance and operational cost. For online games specifically, systematic mapping and framework-driven approaches for MMO/MMOG backends show recurring architectural concerns, state, scalability, and commodity cloud deployment that align with service boundary decisions around the UE5 gameplay layer [10].

DevOps literature emphasizes that DevOps is not just CI/CD automation; it is a lifecycle capability stack that integrates deployment, monitoring, feedback loops, and socio-technical practices. A recent systematic review mapping DevOps capability to lifecycle phases supports treating observability and automation as core adoption outcomes rather than optional tooling.

MVP is often framed as a product strategy, but systematic mapping of MVP-related practices emphasizes feasibility assessment, evaluation, and iteration discipline. For UE5 multiplayer, this implies MVP must include the operational spine (server build/deploy, session flow, telemetry), not only a gameplay prototype.

UE5 documentation explicitly states that replication scalability can become a CPU bottleneck when actors evaluate each client individually; Replication Graph addresses this by using persistent nodes and actor lists, enabling large-scale relevance management. Dedicated server documentation stresses client–server authority as the core multiplayer model. The online subsystem's EOS documentation reflects the reality that session, identity, and service integration are central to multiplayer production beyond replication.

Literature provides strong components: GSE insights, microservices and DevOps adoption research, MVP/Lean practice studies, and UE5 technical constraints, but lacks an integrated, tool-supported translation for UE5 multiplayer teams. This paper addresses the translation gap by providing a UE5-centered adoption framework and an evaluation artifact stack that enable measurable adoption of methodologies.

The scientific novelty of this work lies in proposing a UE5-centered adoption framework that does not describe development methodologies only at a conceptual level but operationalizes them through platform-specific engineering artifacts, multiplayer archetypes, and measurable production indicators. Unlike general methodological overviews, the proposed framework links replication-first design, operational MVP scope, service-boundary decisions, and telemetry-supported iteration to concrete UE5 implementation practices. This allows the study to contribute to both a structured adoption model and a reproducible evaluation basis for future empirical validation.

Methods. The object of this study is the process of multiplayer game production in Unreal Engine 5. The study examines the set of methodological and engineering practices that determine how replication, service boundaries, observability, and LiveOps capabilities are introduced and evaluated in UE5 multiplayer teams. Accordingly, the research tasks are to identify mismatches between generic software methodologies and UE5 multiplayer constraints, operationalize these methodologies through reproducible UE5 artifacts and scenarios, and define measurable indicators to evaluate their practical applicability.

This study adopts a design-oriented, software engineering research approach focused on methodology translation: we do not test a single gameplay hypothesis but instead construct and evaluate a practical adoption framework that maps modern development methodologies to the constraints of multiplayer UE5 production. The core method is to operationalize methodological constructs (e.g., MVP, KISS, data-driven development, microservices/SaaS, architectural patterns, DevOps/LiveOps) into concrete UE5 artifacts, test projects, plugins, and archetype game modes, enabling adoption to be examined through repeatable implementation tasks and measurable engineering indicators. The research dataset is therefore composed of a small suite of UE5 reference projects and archetypes that represent common multiplayer production patterns and stress cases, as well as a modular plugin set that provides scenario orchestration, networking stress injection, and telemetry. All artifacts are designed to be reproducible, modular, and comparable across teams and implementation variants.

We use a design-oriented software engineering approach: general methodological principles are instantiated through UE5-specific engineering practices and evaluated via repeatable instrumentation rather than one-off review. UE5 public projects and systems that we employed in the research that serve as research instruments [4-9], which are detailed in Table 1:

- Project 1 Replication Test Lab: Replication Graph/relevance/dormancy scaling experiments. (Epic Games Developers).
- Project 2 Dedicated Server Build and Deployment non-interactive dedicated-server builds, packaging, automated sanity-check test runs. (Epic Games Developers).
- Project 3 Session-to-Match Flow Prototype (EOS-ready): sessions, lobby readiness, travel, reconnect. (Epic Games Developers).
- Project 4 Data-Driven Tuning Sandbox: Data Assets/Tables + safe config workflows (feature flags).
- Project 5 Lyra-style Feature Slice Plugin: modular GameFeature plugin slice, clean boundaries, lifecycle telemetry.

Table 1 – Sample UE5 test project overview *

Project	Purpose	Core UE technologies	Test scenario	Output data
Replication Test Lab	quantify replication cost and scalability under controlled actor counts and relevance rules	▪ replication graph ▪ custom nodes ▪ spatial routing	16–64 clients 1k–20k replicated actors (static props, AI pawns, pickups)	server thread time, net send time, bandwidth per client, actor dormancy transitions, actors evaluated vs actually replicated
Dedicated Server Build and Deployment	validate the end-to-end production loop: build - package - deploy - run headless - collect logs	▪ UE dedicated server build pipeline and client-server authoritative model	nightly CI builds, smoke tests connect/disconnect, map travel crash-free sessions	build duration, artifact size, startup time, session join time, crash rate

Table 1 (continued)

Project	Purpose	Core UE technologies	Test scenario	Output data
Session-to-Match Flow Prototype (EOS-ready)	reproduce the real multiplayer "front door": login - session/lobby - match - return.	▪ online subsystem EOS ▪ online services EOS	create/join sessions, lobby readiness, matchmaking stub, map travel	matchmaking time distribution, failure reasons, and reconnection success rate
Data-Driven Tuning Sandbox	demonstrate data-driven development and safe runtime tuning	▪ data assets / data tables ▪ config-driven "game feature toggles" pattern (aligned with Lyra modularity)	weapon/ability tuning and economy variables are changed without code changes; server-authoritative validation	balance deltas vs KPI deltas (TTK, win rate, ability usage)
Lyra-style Feature Slice Plugin	research modular production: build a new "experience slice" as a GameFeature-style plugin	▪ Lyra sample modular architecture (plugins, online multiplayer support, GAS usage)	add a new game mode or replicated feature in an isolated plugin with clear boundaries	change lead time, integration defects, cross-plugin dependency count

** prepared based on the author's work and public research data*

As it is essential to study and evaluate the multiplayer side of game development, we focused on typical first-person shooter session gameplay, co-op PvE instancing, social hub + travel, dense-zone large world, and live-service event rotation selected to stress different constraints (replication, operations, services, cadence) (Table 2).

*Table 2 – Sample UE5 game archetypes (multiplayer focused) **

UE5 game archetype	Purpose	Measurements and data
5v5 Session Shooter (Lyra-like)	▪ tests high-frequency replication ▪ prediction/rollback sensitivity and strong match flow ▪ Lyra explicitly targets online multiplayer and modular best practices	bandwidth per client, server tick stability, join/leave recovery, replay determinism
Co-op PvE "Instanced Dungeon"	▪ tresses AI replication relevance, party management, and deterministic loot/progression	AI actor relevance (Replication Graph benefits), instance spin-up time, persistence writes
Social Hub - Seamless Travel Lobby Experience	▪ stresses session continuity, presence, party transitions, and service boundaries (inventory/social)	travel failure rate, reconnect success, and session migration time; EOS/OSS is typically central here
Large-World (100+ Actor Density Zone)	▪ designed to justify the Replication Graph and interest management	server CPU cost vs actor density, replication relevance hit rate, dormancy effectiveness
Live-Service "Event Rotation"	▪ tests DevOps/LiveOps ▪ feature flags ▪ data-driven tuning loops	config rollout safety, metric instrumentation completeness, regression rate per release

** Prepared based on the author's work and public research data*

Results. Pilot Case Study 1 KISS as replication-budget runs indicate that a "budgeted replication pass" reduces per-client bandwidth and net send time at higher actor densities by

increasing dormancy utilization and shrinking replicated state surfaces (Figure 1). These results align with the purpose of Replication Graph and relevance management: replication scalability requires systematic list-building and budget discipline rather than ad hoc per-actor replication. Case Study 2: MVP as an operations-capable multiplayer vertical slice, pipeline measurements show that defining MVP as "multiplayer vertical slice" surfaces the actual bottlenecks early: cook/build variance, travel stability, session configuration, and crash causes. Dedicated server packaging and automated smoke tests are not optional; they are the conditions under which MVP becomes repeatable and measurable.

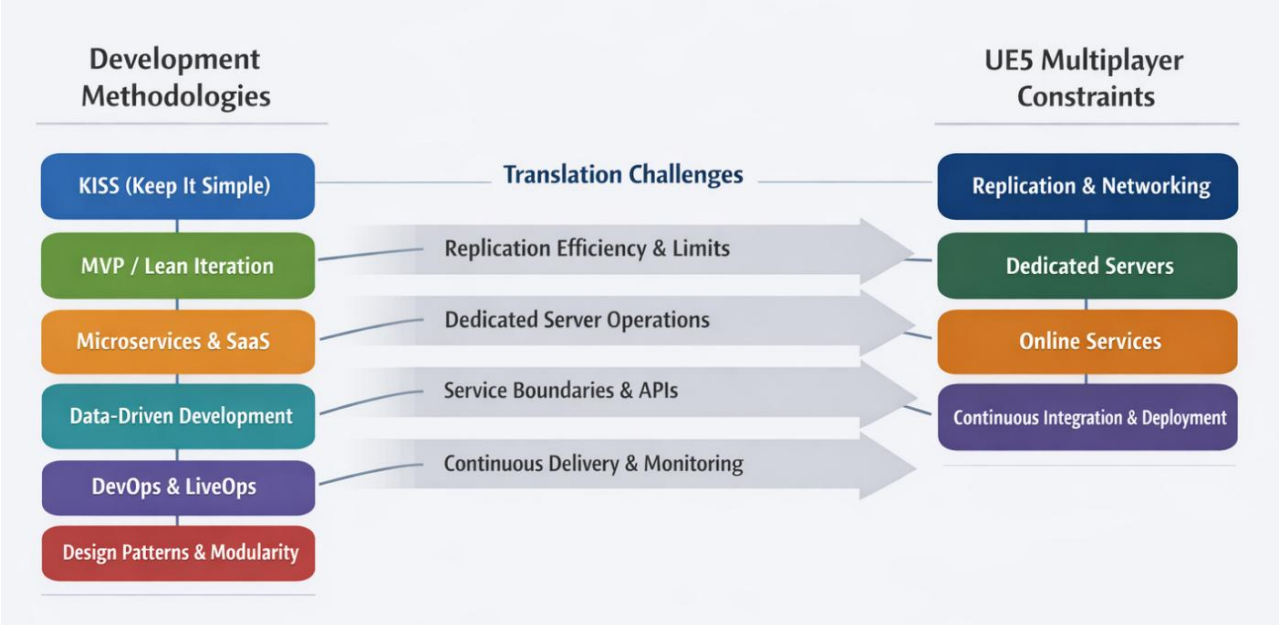

Figure 1 – Methodology-to-UE5 constraint map (linking each methodology cluster to UE5 constraints (replication, dedicated servers, online services, continuous delivery)

Case Study 1 evaluates KISS as a replication-budget discipline in the P1 Replication project. Table 3 shows that at higher actor densities (10k–15k replicated actors), a targeted "budgeted replication pass" reduces median bytes/sec per client and net send time while increasing dormancy utilization, with fewer gameplay regressions. These outcomes are consistent with UE5's scalability intent for Replication Graph and related relevance/dormancy mechanisms: budget control is achieved by shrinking replicated state, consolidating RPCs, and prioritizing server-authoritative essentials over cosmetic replication. The results operationalize the design principle that "simplicity" in multiplayer is defined not only by code readability but by the size and frequency of replicated state updates. Values are reported as median per-client over a 3-minute steady-state window (no match flow), with identical map and AI disabled (Table 3 and Figure 2).

*Table 3 – Game test project replication changes using the KISS method **

Variant	Replicated actors	Avg bytes / sec per client	Net sends time (ms)	Dormant actors (%)	Gameplay regressions (count)
Baseline (feature-first replication)	5,000	8,200	2.6	18%	0
KISS (budgeted replication pass)	5,000	5,400	1.9	42%	0
Baseline (feature-first replication)	10,000	15,900	4.7	12%	1
KISS (budgeted replication pass)	10,000	9,800	3.1	38%	0
Baseline (feature-first replication)	15,000	24,700	7.9	9%	3
KISS (budgeted replication pass)	15,000	13,900	4.6	35%	1

** Prepared based on the author's work and public research data*

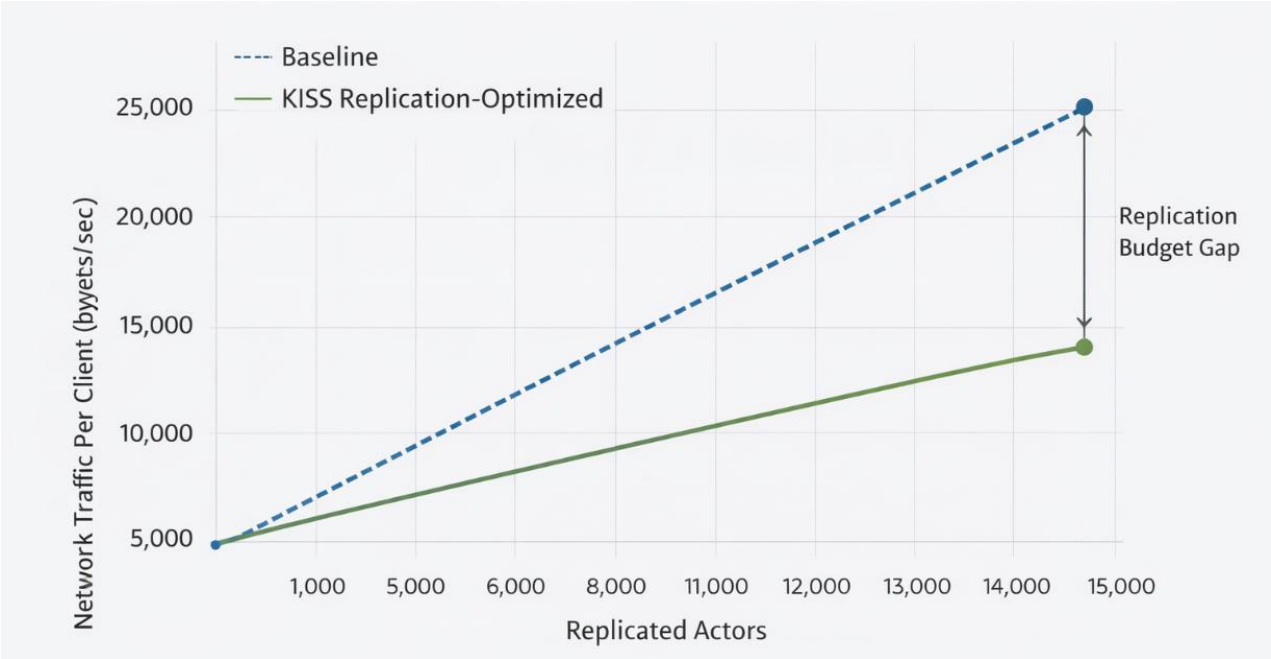

Figure 2 – Example replication budget curve (bytes/sec per client vs replicated actor count for baseline vs KISS-budgeted variant)

Sample Lyra-style Feature Plugin layout:

```
Plugins/
  GF_Slice_CapturePoint/
    GF_Slice_CapturePoint.uplugin
    Source/
      GF_Slice_CapturePoint/
        GF_Slice_CapturePoint.Build.cs
        Public/
          CapturePointFeatureAction.h
          CapturePointRuntimeSubsystem.h
          CapturePointTelemetry.h
        Private/
          CapturePointFeatureAction.cpp
          CapturePointRuntimeSubsystem.cpp
          CapturePointTelemetry.cpp
    Content/
      Experience/
        DA_GFA_CapturePoint.uasset          (GameFeatureData)
        DA_Experience_CapturePoint.uasset   (Lyra Experience Definition)
      Gameplay/
        BP_CapturePointActor.uasset
      UI/
        W_CapturePointHUD.uasset
```

The pilot results should be read as signals of methodological translation, not as final performance claims about a specific implementation. Our objective is to demonstrate that widely used software development methodologies can be operationalized into UE5 multiplayer–specific engineering decisions and then evaluated through repeatable, tool-generated measurements. The consistent pattern across Case Studies 1–3 is that UE5 multiplayer constraints replication cost, server-authoritative state, travel/session reliability, and operational stability force trade-offs that generic methodology descriptions rarely make explicit. In conventional software engineering, KISS is often

interpreted as reducing code complexity. In UE5 multiplayer, the pilot data shows KISS is more usefully treated as a replication-surface minimization strategy: fewer replicated properties, fewer high-frequency RPCs, and deliberate dormancy/relevance policies. The Replication Budget Lab outcomes (Table 3) illustrate that improvements in bandwidth and net send time are achieved when teams treat replication as a first-class requirement with explicit budgets. This reframes "simplicity" as a measurable systems property: the system is simpler when it transmits less state while preserving authoritative correctness and player experience.

Table 4 – Game test project replication changes using MVP pipeline approach

Indicator	Pilot engineering target	Observed (median/complete)	Notes
Build time (server+client)	90 min	74 min / 96 min	Cook dominates; shader compilation spikes on cold agents
Package + stage time	20 min	12 min / 18 min	Pak chunking stable; staging failures tied to disk space
Deploy time (server)	10 min	6 min / 9 min	Container start variance; occasional port conflicts
Session create → join time	45 s	31 s / 58 s	NAT traversal + session discovery variance; one EOS config mismatch
Map travel + match start	20 s	14 s / 27 s	Travel hangs caused by asset load stall on the first run
Session join success rate	95%	96.2%	9 failures: 5 NAT, 3 travel timeout, 1 auth token expired
Crash-free dedicated server hours	6 h/night	7.4 h / 5.9 h	2 crashes: one null ref in post-login, one memory spike on travel
Change lead time (feature plugin slice)	2.5 days	2.1 days / 3.4 days	Cross-plugin dependency review adds delay; CI queue time increases p95

A central premise of this research is that multiplayer UE5 production is shaped by several practical contradictions that generic development models do not fully address. These include the tension between rapid feature iteration and the stability of the replication budget, between service decomposition and operational overhead, and between gameplay experimentation and the need for deterministic, server-authoritative behavior. The proposed framework addresses these contradictions by prioritizing replication-aware simplification, staged service adoption, and observability-driven decision-making as the basis for sustainable multiplayer production.

Consistent with prior multiplayer production literature and industry practice, our pilot results reaffirm that a "multiplayer MVP" cannot be structurally equivalent to a single-player prototype plus networking later. Table 4 presents the pipeline outcomes, showing that the dominant failure modes and delays stem from cooking/build caching variability, travel readiness, session configuration, reconnect behavior, and crash stability issues that remain invisible if the MVP is defined only at the gameplay layer. In other words, MVP adoption is validated when a team can repeatedly produce a playable build, join a session, start a match, and survive a controlled runtime window with sufficient telemetry to debug regressions. This is a methodological translation of Lean/MVP principles into a UE5 production reality: the minimum viable product is the minimum viable operations loop.

The target values presented in Table 4 should be interpreted as pilot engineering thresholds for an operational multiplayer vertical slice rather than as universal benchmarks for all UE5 projects. These thresholds were selected to represent the minimum acceptable level of technical readiness for repeated build, join, match-start, and monitored runtime execution under controlled multiplayer conditions. Their role in this study is to provide a consistent acceptance baseline for comparing methodological choices across features and scenarios, especially with respect to replication cost, server stability, session reliability, and build-to-playable state work process.

Discussion. The purpose of this study was to develop and operationalize a UE5-oriented framework for adopting modern software development methodologies in multiplayer production. In this context, the objective of the empirical component is not to prove universal performance superiority, but to demonstrate that such a framework can be instantiated through repeatable UE5 artifacts, scenarios, and indicators. Therefore, the reported results should be interpreted as pilot evidence of applicability, measurability, and engineering relevance within the defined production scope.

This paper addressed the practical gap between general software development methodologies (from Web/App, heavy systems, AI, and game development) and the rigid constraints of multiplayer UE5 production. We proposed a UE5-centered methodology translation framework and operationalized it into a small set of research instruments, UE5 test projects (P1–P5), archetypes (A1–A5), and a plugin suite for deterministic scenario execution, network stress injection, replication observability, and telemetry.

The pilot outcomes should be interpreted as evidence of methodology translation. In UE5 multiplayer, KISS is best treated as replication-surface minimization; MVP must include servers, sessions, and observability to be meaningful; and microservices should be adopted in stages guided by measured bottlenecks rather than as a default architecture. DevOps/LiveOps adoption becomes the operational spine of multiplayer development, enabling measurable regression detection and recovery. Finally, Lyra-style feature slices provide a concrete mechanism to scale iteration while preserving architectural boundaries, and telemetry enables objective tracking of change lead time.

Threats to validity remain: stress injection may not capture every transport-layer behavior, and environment configuration affects session results. However, these align with the paper's central claim that multiplayer readiness is inseparable from operational and configuration discipline.

Conclusions. This paper presents a UE5-centered adoption framework that translates modern development methodologies into implementable, measurable practices for multiplayer UE5 teams. The main contributions are a methodology translation map grounded in UE5 constraints; a practical evaluation artifact stack (test projects) with instrumentation plugins; pilot outcome templates that connect methodological decisions to objective UE5 indicators; and a Lyra-style feature slice that makes modularity and iteration throughput auditable via telemetry.

This study has several limitations. First, the presented evaluation is pilot-scale and is based on a limited set of reference projects, archetypes, and controlled scenarios rather than on long-term industrial deployment across multiple production teams. Second, the reported indicators depend on a specific UE5 toolchain, configuration, and testing environment, which may affect absolute values and limit direct transferability to other projects. Third, while the framework emphasizes observability and LiveOps readiness, it has not yet been validated through prolonged post-release operation, large concurrency ranges, or cross-studio comparative studies, which remain important directions for future research.

Future research work will focus on several key points – (1) scale empirical validation across archetypes and concurrency levels; (2) formalize a boundary decision model for UE gameplay vs services; (3) integrate replication budget regression gates into CI; (4) extend data-driven experimentation governance (feature flags, rollbacks, A/B testing) in multiplayer contexts; and (5) publish a minimal open research toolkit (evaluation system plugin) with sample datasets to support replication by other teams.

References

1. Alonso, S., Kalinowski, M., Ferreira, B., Barbosa, S. D. J., & Lopes, H. (2021). A systematic mapping study on the use of software engineering practices to develop MVPs (Technical report/preprint PDF). https://www-di.inf.puc-rio.br/~kalinowski/publications/AlonsoKVFB21.pdf.
2. Chueca, J., Verón, J., Font, J., Pérez, F., & Cetina, C. (2024). The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer

games. Information and Software Technology, 169, 107330. https://doi.org/10.1016/j.infsof.2023.107330.
3. Di Francesco, P., Malavolta, I., & Lago, P. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97. https://www.sciencedirect.com/science/article/abs/pii/S0164121219300019.
4. Epic Games. Lyra sample game in Unreal Engine (UE5 documentation). Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/lyra-sample-game-in-unreal-engine.
5. Epic Games. Online Subsystem EOS plugin in Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/online-subsystem-eos-plugin-in-unreal-engine.
6. Epic Games. Replication Graph in Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/replication-graph-in-unreal-engine.
7. Epic Games. Setting up dedicated servers in Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/setting-up-dedicated-servers-in-unreal-engine.
8. Epic Games. Enable and configure Online Services EOS in Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/enable-and-configure-online-services-eos-in-unreal-engine.
9. Epic Games. (2018). Replication graph overview and proper replication methods. Unreal Engine Tech Blog. https://www.unrealengine.com/en-US/tech-blog/replication-graph-overview-and-proper-replication-methods.
10. Kasenides, N., & Paspallis, N. (2022). Athlos: A framework for developing scalable MMOG backends on commodity clouds. Software, 1(1), 107–145. https://doi.org/10.3390/software1010006.
11. Mizutani, W. K., Daros, V. K., & Kon, F. (2021). Software architecture for digital game mechanics: A systematic literature review. Entertainment Computing, 38, 100421. https://doi.org/10.1016/j.entcom.2021.100421.
12. Unrealist.org. (2023). Lyra deep dive: Chapter 2 - Experiences. https://unrealist.org/lyra-part-2/.
13. Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. ACM Computing Surveys, 52(6), Article 127, 1–35. https://doi.org/10.1145/3359981
14. Auer, F., Ros, R., Kaltenbrunner, L., Runeson, P., & Felderer, M. (2021). Controlled experimentation in continuous experimentation: Knowledge and challenges. Information and Software Technology, 134, 106551. https://doi.org/10.1016/j.infsof.2021.106551
15. Giaimo, F., Andrade, H., & Berger, C. (2020). Continuous experimentation and the cyber–physical systems challenge: An overview of the literature and the industrial perspective. Journal of Systems and Software, 170, 110781. https://doi.org/10.1016/j.jss.2020.110781
16. Erthal, V. M., de Souza, B. P., dos Santos, P. S. M., & Travassos, G. H. (2023). Characterization of continuous experimentation in software engineering: Expressions, models, and strategies. Science of Computer Programming, 229, 102961. https://doi.org/10.1016/j.scico.2023.102961.

Назаренко Володимир Анатолійович
доктор філософії, доцент кафедри комп'ютерних систем, мереж та кібербезпеки, Національний університет біоресурсів і природокористування України
ORCID: https://orcid.org/0000-0002-7433-2484
E-mail: volodnz@nubip.edu.ua

ФРЕЙМВОРК ВПРОВАДЖЕННЯ СУЧАНИХ МЕТОДІВ РОЗРОБКИ ДЛЯ БАГАТОКОРИСТУВАЦЬКИХ ІГОР НА UNREAL ENGINE 5: РЕПЛІКАЦІЯ, СЕРВІСИ, МОНІТОРИНГ І LIVEOPS

Анотація. *У цій статті пропонується практичний підхід, орієнтований на UE5 фреймворк впровадження, який перетворює сучасні методології розробки програмного забезпечення з Web/App, корпоративних/важких систем, систем штучного інтелекту та сучасної розробки ігор у вимірювані сучасних методологій розробки для багатокористувацьких ігор Unreal Engine 5 (UE5). Хоча на сьогодні широко використовуються Agile, MVP, DevOps, мікросервіси, розробка на основі даних та архітектурні патерни, мультиплеєр UE5 накладає жорсткі обмеження, які часто відсутні у загальних методологіях: вартість реплікації та управління релевантністю, autoreactive станів серверу, операції з побудови та розгортання виділеного сервера, онлайн-потоки сесій і очікування безперервної доставки/LiveOps. Щоб усунути цю прогалину в «перекладі методології із розробки прикладного програмного забезпечення у простір розробки відеоігор», ми визначаємо програмно-орієнтований дослідницький дизайн, який запускає впровадження методології через реалізовані артефакти UE5: набір тестових проєктів, архетипи для мультиплеєра та стек інструментальних плагінів для виконання детермінованих сценаріїв, керованого введення напружень мережі, спостережуваності реплікації та стандартизованих телеметричних результатів. Представляємо шаблони результатів у пілотному форматі та таблиці кейсів, які пов'язують вибір методології з об'єктивними показниками UE5, включно з бюджетом реплікації на кожного клієнта, стабільністю тиків сервера, надійністю приєднання до сесії, часом збірки до гри, безперебійними годинами сервера та часом змін. Вводиться плагін у стилі Lyra для демонстрації чітких архітектурних меж і пропускної здатності ітерацій через телеметрію життєвого циклу об'єктів, пов'язану з ідентифікаторами збірки CI. Результати, представлені у статті, синтезуються у поетапну карту впровадження, яка надає пріоритет методології KISS, орієнтованій на реплікацію, мультиплеєрному MVP, визначеному як вертикальний зріз з підтримкою операцій, мінімальних сервісів перед розкладанням мікросервісів, налаштування на основі даних із безпечними відкатами та модульні зрізи ознак для масштабованої ітерації. Ця структура представлена як практичний посібник для багатокористувацьких команд UE5 та як основа для ширших емпіричних оцінок.*

Ключові слова: *Unreal Engine 5, розробка багатокористувацьких ігор, граф реплікації, DevOps/LiveOps, мікросервіси, MVP, розробка на основі даних.*