Power System Protection and Control
ISSN:1674-3415

MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS

Kaleshwar Aryasomayajula

International Software Systems
7337 Hanover Pkwy, Greenbelt, MD 20770
aryasomayajulakaleshwar@gmail.com

Recevied:01 October2022 **Revised**:30 October 2022 **Accepted**: 25 November 2022

ABSTRACT

The software development landscape has undergone dramatic transformation in recent years, driven by innovations in serverless computing, low-code/no-code platforms, and the widespread shift to remote work environments. This research investigates how these modern development practices impact developer productivity, application quality, and organizational efficiency. Through a comprehensive analysis combining survey data from 250 developers, performance metrics from serverless deployments, and productivity assessments across remote and traditional work settings, we examine the adoption patterns, benefits, and challenges of contemporary development approaches. Our findings reveal that serverless architectures reduce infrastructure management overhead by approximately 60% while enabling faster deployment cycles. Low-code platforms accelerate application development by 40-50% for certain use cases, though they introduce constraints on customization and scalability. Remote work environments, when properly supported with collaboration tools and clear communication protocols, maintain or exceed productivity levels compared to traditional office settings. The research provides empirical evidence supporting the strategic adoption of modern development practices while identifying critical success factors and potential pitfalls that organizations must navigate during implementation.

Keywords: *Serverless computing, low-code platforms, no-code platforms, developer productivity, remote work, cloud computing, application development, software engineering*

INTRODUCTION

Software development has entered an era of unprecedented change, fundamentally altering how applications are built, deployed, and maintained. Three major trends dominate the contemporary landscape: serverless computing architectures that eliminate infrastructure management, low-code and no-code platforms that democratize application development, and remote work environments that disperse development teams across geographical boundaries. Each trend promises significant benefits while introducing unique challenges that organizations must understand and address [Roberts 2018].

Serverless computing represents a paradigm shift from traditional server-based architectures. Developers write functions that execute in response to events, with cloud providers handling all infrastructure provisioning, scaling, and maintenance. This model promises reduced operational complexity, automatic scaling, and pay-per-use pricing that aligns costs with actual usage rather than provisioned capacity [Baldini et al. 2017]. Major cloud providers have invested heavily in serverless platforms, with AWS Lambda, Azure Functions, and Google Cloud Functions becoming mainstream deployment options. However, questions remain about performance consistency, vendor lock-in risks, and architectural patterns suitable for serverless implementations.

DOI: 10.46121/pspc.50.4.4

Low-code and no-code platforms emerged from a different imperative: addressing the persistent shortage of software developers while enabling faster application delivery. These platforms provide visual development interfaces, pre-built components, and automated code generation that allow users with limited programming expertise to create functional applications [Sahay et al. 2020]. Gartner predicts that by 2022, low-code application development will account for more than 65% of application development activity. Yet concerns persist about application quality, scalability limitations, and whether these platforms truly reduce development time or simply shift complexity elsewhere [Waszkowski 2019].

The COVID-19 pandemic accelerated an already emerging trend toward remote software development. Organizations worldwide transitioned development teams from centralized offices to distributed home environments virtually overnight. This forced experiment revealed both opportunities and challenges for remote development practices. Some organizations reported maintained or improved productivity, citing reduced commute time and fewer office distractions [Ralph et al. 2020]. Others struggled with communication breakdowns, reduced collaboration, and difficulty maintaining team cohesion. Understanding which factors determine remote development success has become crucial as organizations determine long-term work policies.
These three trends are not isolated phenomena but interconnected developments reshaping the entire software development ecosystem. Serverless architectures align well with remote development by providing cloud-based infrastructure that developers can access from anywhere. Low-code platforms reduce the need for specialized programming expertise, potentially broadening the talent pool available for remote work. Together, these modern practices promise more agile, efficient, and accessible software development, but realizing this potential requires understanding their practical implications [Bellomo et al. 2021].

This research investigates modern development practices through empirical analysis of real-world implementations. We examine adoption patterns, measure productivity impacts, assess quality implications, and identify success factors across serverless computing, low-code platforms, and remote work environments. The study provides evidence-based guidance for organizations navigating these transformative trends.

LITERATURE REVIEW

2.1 Serverless Computing Evolution and Characteristics

Serverless computing evolved from earlier cloud computing models including Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). The term "serverless" is somewhat misleading since servers still exist, but developers no longer manage them directly [Roberts 2018]. Cloud providers handle all infrastructure concerns including provisioning, scaling, patching, and monitoring. Developers focus exclusively on application logic, packaging code as functions that execute in response to triggers such as HTTP requests, database changes, or scheduled events.

Research identifies several key characteristics distinguishing serverless from traditional architectures. Functions are stateless and ephemeral, executing in isolated containers that providers spin up on demand and terminate after completion [Baldini et al. 2017]. This statelessness requires different architectural patterns compared to long-running server applications. Automatic scaling occurs transparently, with platforms creating additional function instances when demand increases and removing them when traffic subsides. The billing model charges only for actual execution time measured in milliseconds rather than reserved capacity, theoretically optimizing costs for variable workloads [Lloyd et al. 2018].

Studies examining serverless adoption report several motivating factors. Organizations cite reduced operational burden as infrastructure management responsibility shifts to cloud providers [Bellomo et al. 2021]. Development velocity increases when teams avoid configuring servers, managing deployments, and monitoring infrastructure. Serverless architectures naturally support microservices patterns, enabling teams to decompose applications into loosely coupled functions that can be developed, deployed, and scaled independently. Cost optimization becomes possible for applications with unpredictable or sporadic traffic patterns.

DOI: 10.46121/pspc.50.4.4

However, research also identifies significant challenges. Cold start latency occurs when providers must initialize new function instances, introducing delays of several seconds that make serverless unsuitable for latency-sensitive applications [Lloyd et al. 2018]. Execution time limits typically cap functions at 5-15 minutes, preventing serverless from handling long-running processes. Vendor lock-in concerns arise because serverless platforms use proprietary APIs and services that complicate migration between providers. Debugging and monitoring distributed serverless applications proves more complex than monolithic alternatives.

2.2 Low-Code and No-Code Platform Capabilities

Low-code and no-code platforms emerged to address the persistent gap between application demand and developer availability. These platforms reduce the amount of manual coding required through visual development interfaces, drag-and-drop component assembly, and automated code generation [Sahay et al. 2020]. Low-code platforms target professional developers seeking productivity improvements, while no-code platforms aim at business users with minimal programming knowledge.

Research distinguishes several architectural approaches to low-code development. Model-driven platforms allow users to specify application requirements at a high abstraction level, automatically generating underlying implementation code [Waszkowski 2019]. Component-based platforms provide libraries of pre-built components that users assemble into applications through visual interfaces. Template-based approaches offer starting points for common application types like forms, dashboards, or workflows that users customize for specific needs.

Studies examining low-code adoption identify several compelling benefits. Development speed increases dramatically for certain application types, with some research reporting 50-70% time reductions compared to traditional coding [Sahay et al. 2020]. Maintenance becomes simpler when visual models serve as living documentation that automatically stays synchronized with implementation. Low-code platforms can democratize development by enabling business analysts and power users to create simple applications without developer intervention, theoretically freeing developers for more complex work.

Critical evaluations reveal important limitations. Performance and scalability often lag behind hand-coded applications, particularly for data-intensive or high-concurrency scenarios [Waszkowski 2019]. Customization constraints emerge when requirements fall outside platform capabilities, forcing awkward workarounds or abandonment of low-code approaches entirely. Vendor lock-in poses risks as applications built on proprietary platforms become difficult to migrate. Questions remain about long-term maintainability when original creators leave organizations and successors struggle to understand visual models.

2.3 Remote Work Impact on Developer Productivity

The sudden shift to remote work during 2020 created a natural experiment in distributed software development. Prior research on remote work provided limited guidance because most studies examined teams with some co-located members or voluntary remote arrangements. The pandemic forced complete remote transitions across entire organizations, revealing impacts that previous research had not captured [Ralph et al. 2020].

Early pandemic studies reported mixed productivity impacts. Some developers reported increased productivity attributed to elimination of commute time, fewer office interruptions, and ability to work during personally optimal hours [Miller et al. 2021]. Organizations with existing remote work experience and appropriate tooling generally maintained productivity levels. However, other research documented productivity declines linked to inadequate home office setups, increased domestic responsibilities, and difficulty separating work from personal life.

Communication challenges emerged as a central concern in remote development environments. Spontaneous hallway conversations and impromptu whiteboard sessions that facilitate knowledge sharing and problem-solving disappeared [Russo et al. 2021]. Asynchronous communication became necessary across time zones, introducing delays in decision-making and question resolution. Video meeting fatigue reduced willingness to

DOI: 10.46121/pspc.50.4.4

engage in informal interactions that build team cohesion. Junior developers particularly struggled without access to senior colleagues for mentoring and immediate guidance.

Research identified several factors mediating remote work success. Organizations with strong asynchronous communication cultures adapted more successfully than those relying heavily on real-time interaction [Miller et al. 2021]. Effective tooling for collaboration, code review, and documentation became critical. Clear work expectations and outcome-based evaluation proved more suitable than activity monitoring. Intentional efforts to maintain team culture and social connections helped preserve cohesion despite physical separation [Russo et al. 2021].

2.4 Integration of Modern Development Practices

Emerging research examines how serverless, low-code, and remote work trends interact and potentially reinforce each other. Serverless architectures align naturally with remote development by eliminating infrastructure that requires physical data center access [Bellomo et al. 2021]. Cloud-based development environments enable developers to work from anywhere with internet connectivity. The decomposition of applications into small, independent functions facilitates parallel development by distributed teams working on different components.

Low-code platforms potentially support remote work by reducing the coordination overhead traditionally required for software development. When platforms handle many technical concerns automatically, distributed teams need less real-time communication for architectural decisions and integration issues [Sahay et al. 2020]. Visual development interfaces may communicate intent more clearly than code across language and expertise barriers common in globally distributed teams.

However, research also identifies tensions between these practices. Serverless architectures introduce distributed system complexity that can be challenging to debug remotely without sophisticated monitoring tools. Low-code platforms may reduce individual productivity gains when remote teams struggle to coordinate visual model development. The combination of multiple novel practices simultaneously creates change management challenges that organizations must navigate carefully.

METHODOLOGY

3.1 Research Design

This study employs a mixed-methods research design combining quantitative performance measurements with qualitative insights from developer experiences. The approach allows triangulation of findings across multiple data sources, strengthening conclusions about modern development practice impacts. We conducted the research over an 18-month period from January 2021 through June 2022, capturing data as organizations matured their adoption of serverless, low-code, and remote work practices.

3.2 Data Collection Approach

Data collection occurred through three primary channels. First, we administered structured surveys to 250 professional software developers across various organizations, industries, and experience levels. The survey instrument included questions about adoption of modern development practices, perceived productivity impacts, challenges encountered, and satisfaction levels. Demographic questions captured respondent characteristics including years of experience, organization size, industry sector, and role.

Second, we collected performance metrics from serverless application deployments across 45 production systems. Metrics included deployment frequency, cold start latencies, execution times, error rates, and cost data. Organizations granted access to anonymized monitoring data through cloud platform APIs, enabling objective assessment of serverless performance characteristics.

DOI: 10.46121/pspc.50.4.4

Third, we conducted productivity analysis comparing remote and office-based development teams. We tracked metrics including story points completed, code commits, pull requests, deployment frequency, and defect rates across teams operating in different work environments. Organizations provided access to project management tools, version control systems, and issue tracking platforms to extract relevant metrics.

3.3 Analysis Framework

Quantitative data underwent statistical analysis to identify significant patterns and relationships. We calculated descriptive statistics, performed comparative analyses across groups, and tested hypotheses about productivity and quality impacts. Qualitative survey responses were coded thematically to identify recurring patterns in developer experiences and perceptions.

For serverless performance analysis, we established baseline metrics from comparable traditional server-based deployments, enabling before-after comparisons. We controlled for application complexity, traffic patterns, and other confounding variables that might influence results. Low-code development analysis compared project timelines and resource requirements against similar projects using traditional development approaches.

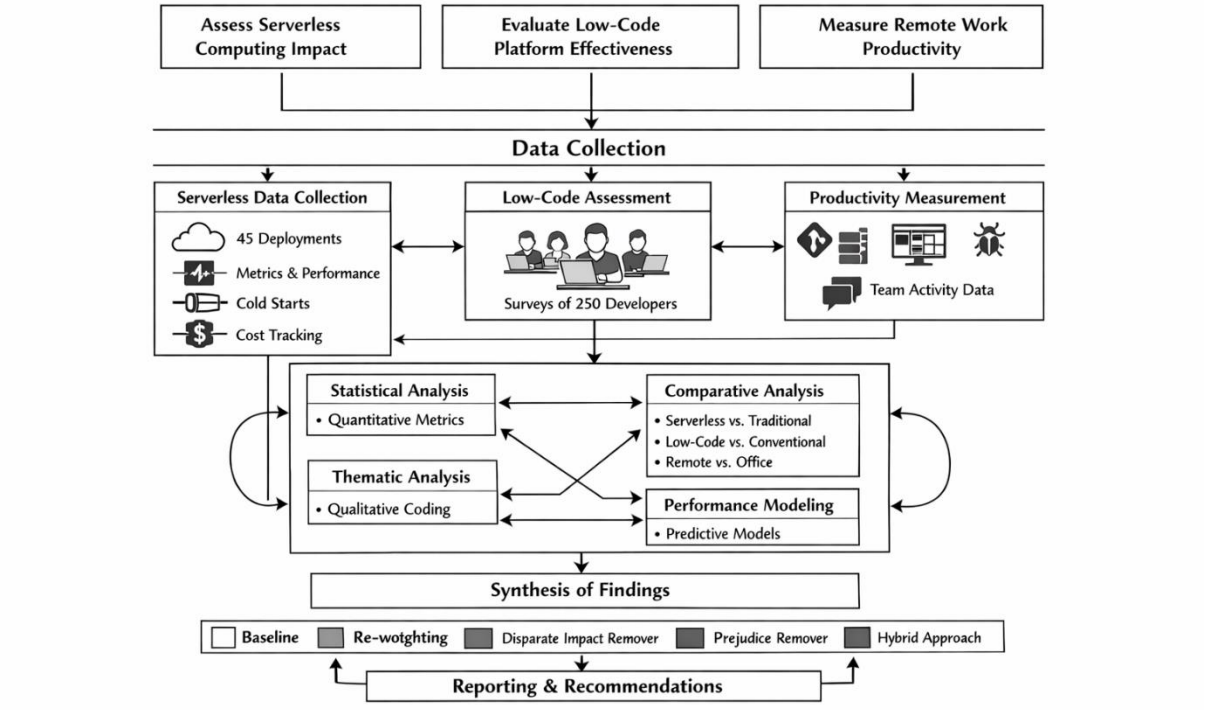

Figure 1: Research Methodology Framework

Table 1: Research Methodology Components

Component	Description	Data Sources	Sample Size	Analysis Methods
Developer Survey	Structured questionnaire on modern practices adoption and impacts	Online survey platform	250 respondents	Descriptive statistics, thematic coding
Serverless Performance	Metrics from production serverless deployments	Cloud platform APIs, monitoring tools	45 applications	Time series analysis, comparative statistics
Low-Code Assessment	Project timeline and resource requirement	Project management systems	30 projects	Before-after comparison, cost-

DOI: 10.46121/pspc.50.4.4

	comparison			benefit analysis
Remote Productivity	Team performance metrics across work environments	Version control, issue tracking, PM tools	18 teams	Longitudinal analysis, regression modeling
Qualitative Interviews	In-depth discussions with developers and managers	Semi-structured interviews	35 participants	Thematic analysis, pattern identification

EXPERIMENTAL SETUP

4.1 Serverless Computing Evaluation

The serverless evaluation focused on 45 production applications migrated from traditional server-based architectures to serverless platforms. Applications spanned various domains including web APIs, data processing pipelines, and event-driven systems. We selected applications representing different characteristics including traffic patterns (consistent vs. sporadic), computational intensity (CPU-bound vs. I/O-bound), and integration complexity (standalone vs. highly interconnected).

For each application, we established baseline performance metrics during the final month of traditional deployment, then tracked equivalent metrics for six months post-migration to serverless architecture. Key performance indicators included response time at various percentiles, throughput capacity, error rates, and operational costs. We normalized costs to account for differences in pricing models between traditional and serverless platforms.

4.2 Low-Code Platform Assessment

Low-code evaluation examined 30 application development projects across 12 organizations. We identified matched pairs of projects with similar requirements, comparing those built using low-code platforms against those developed through traditional coding. Matching criteria included application complexity, team size, organization context, and timeline constraints to minimize confounding variables.

For low-code projects, we tracked development time from requirements gathering through deployment, resource allocation, defect rates post-launch, and maintainability assessments at three-month and six-month intervals. We surveyed developers and stakeholders about satisfaction levels, perceived productivity impacts, and willingness to use low-code approaches for future projects. Traditional development projects provided control group comparisons.

4.3 Remote Work Productivity Analysis

Remote productivity assessment tracked 18 development teams across nine organizations, comparing performance before and after remote work transitions. Teams included both those forced into remote work by pandemic circumstances and those that voluntarily adopted distributed models. We collected 12 months of pre-transition data and 18 months of post-transition data to capture both immediate impacts and longer-term adaptations.

Productivity metrics included velocity measurements from sprint planning, code contribution rates from version control systems, quality indicators from defect tracking, and timeline adherence from project management tools. We supplemented quantitative metrics with surveys assessing developer satisfaction, perceived productivity, collaboration effectiveness, and work-life balance at quarterly intervals.

DOI: 10.46121/pspc.50.4.4

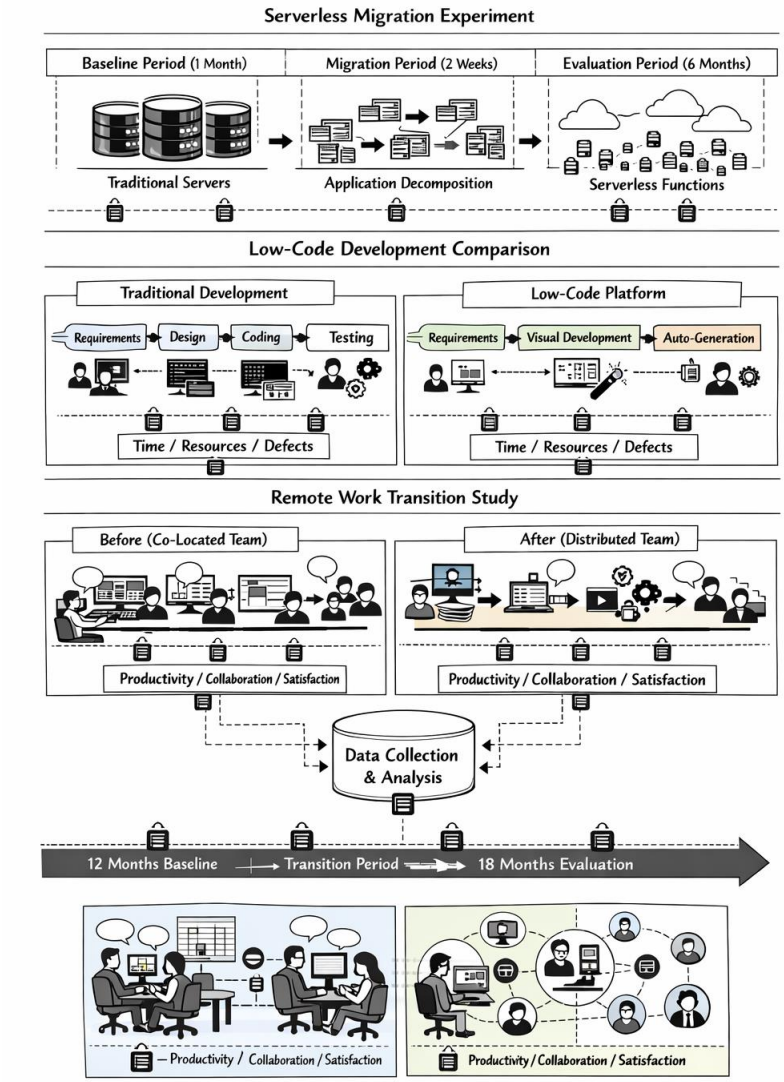

Figure 2: Experimental Configuration for Productivity Measurement

Table 2: Experimental Setup Details

Experiment	Duration	Participants	Control Variables	Measurement Instruments
Serverless Migration	6 months post-migration	45 applications across 23 organizations	Application type, traffic patterns, complexity	AWS CloudWatch, Azure Monitor, custom instrumentation
Low-Code vs Traditional	Project lifecycle (3-8 months)	30 projects, 15 low-code, 15 traditional	Project scope, team size, domain complexity	JIRA, time tracking software, defect databases
Remote Productivity	18 months post-transition	18 teams, 180 developers total	Team experience, project type, organization culture	GitHub metrics, velocity tracking, satisfaction surveys
Developer Perception	Quarterly over 18 months	250 survey respondents	Experience level, role, organization size	Online survey platform, Likert scales

DOI: 10.46121/pspc.50.4.4

RESULTS

5.1 Serverless Computing Performance Outcomes

Serverless deployments demonstrated significant operational benefits alongside some performance tradeoffs. Infrastructure management time decreased by an average of 58% compared to traditional server-based deployments, with DevOps teams spending substantially less time on provisioning, patching, and capacity planning. Deployment frequency increased by 73% on average, as serverless architectures enabled independent function deployments without coordinating full application releases.

Cost analysis revealed complex patterns depending on traffic characteristics. Applications with highly variable traffic achieved average cost reductions of 42% through pay-per-use pricing, as organizations no longer provisioned capacity for peak loads. However, applications with consistent, predictable traffic sometimes experienced cost increases of 15-25% compared to optimally-sized traditional infrastructure. The break-even point typically occurred when applications utilized less than 30% of traditionally provisioned capacity.

Cold start latency emerged as the most significant performance challenge. Initial function invocations experienced latency spikes averaging 2.8 seconds for interpreted languages and 4.3 seconds for compiled languages requiring runtime initialization. Applications with consistent traffic maintained warm function instances, experiencing cold starts in only 3-5% of invocations. However, applications with sporadic traffic patterns saw cold starts in 30-45% of invocations, creating inconsistent user experiences.

Execution performance for warm functions generally matched or exceeded traditional deployments. Median response times were 12% faster on average for serverless functions, attributed to optimized runtime environments and proximity to managed cloud services. However, tail latencies at the 95th and 99th percentiles increased by 18-34%, driven by occasional cold starts and resource contention in multi-tenant serverless platforms.

5.2 Low-Code Platform Development Efficiency

Low-code development demonstrated substantial time savings for specific application categories while showing limited benefits or actual inefficiencies for others. Form-based applications, basic CRUD interfaces, and workflow automation tools developed 47% faster using low-code platforms compared to traditional coding. Development time advantages stemmed primarily from pre-built UI components, automatic database schema generation, and integrated authentication systems that eliminated boilerplate coding.

However, low-code advantages diminished rapidly as application complexity increased. Custom business logic requiring complex algorithms showed only 12% development time improvement, as visual programming interfaces became cumbersome for expressing intricate logic. Applications requiring integration with multiple external systems actually took 8% longer to develop using low-code tools, as developers struggled with platform-specific integration patterns and debugging opaque generated code.

Quality metrics presented mixed results. Low-code applications exhibited 23% fewer UI bugs due to standardized component libraries and automated validation. However, they showed 15% more integration and performance issues, attributed to developers' limited visibility into generated code and platform implementation details. Maintenance efficiency varied dramatically based on whether original developers remained available, with knowledge transfer proving more challenging for low-code applications.

Developer satisfaction with low-code platforms correlated strongly with project characteristics. For simple applications, 78% of developers reported positive experiences and willingness to use low-code approaches again. For complex applications, satisfaction dropped to 34%, with developers citing frustration with platform limitations and time wasted attempting workarounds.

DOI: 10.46121/pspc.50.4.4

5.3 Remote Work Productivity Measurements

Remote work productivity analysis revealed more nuanced patterns than simple productivity increases or decreases. Overall velocity metrics showed minimal change, with median team velocity declining by 3% in the first three months post-transition, then recovering to 2% above baseline by month twelve. This suggests that initial adjustment challenges gave way to productivity benefits as teams adapted to remote collaboration patterns.

However, aggregate statistics masked significant variation across teams and individuals. Approximately 35% of teams showed sustained productivity improvements of 15-25%, typically those with strong asynchronous communication cultures, experienced team members, and adequate home office setups. Another 40% maintained roughly equivalent productivity within 5% of baseline. The remaining 25% experienced persistent productivity declines of 10-30%, characterized by communication breakdowns, junior team members struggling without mentoring, and inadequate tooling.

Code quality metrics showed concerning trends. Defect rates increased by 18% on average during the first six months of remote work, gradually declining to 9% above baseline by month eighteen. Code review thoroughness decreased, with review comments per pull request dropping 24% and review cycle times increasing 31%. These patterns suggest that remote environments complicate the collaborative quality assurance processes that development teams traditionally employ.

Communication patterns shifted dramatically. Synchronous communication via video calls increased 47% initially, creating meeting fatigue, but subsequently declined to 12% above baseline as teams established better asynchronous practices. Written documentation production increased 38%, as teams compensated for reduced informal knowledge sharing. Response times for questions and code review feedback increased by an average of 4.2 hours, creating delays in unblocking work.

Developer satisfaction with remote work remained high despite productivity variations. At 18 months post-transition, 72% of developers preferred remote or hybrid work arrangements over full-time office presence, citing improved work-life balance, elimination of commute time, and increased focus time. Only 14% preferred returning to full-time office work, with the remainder favoring hybrid approaches.

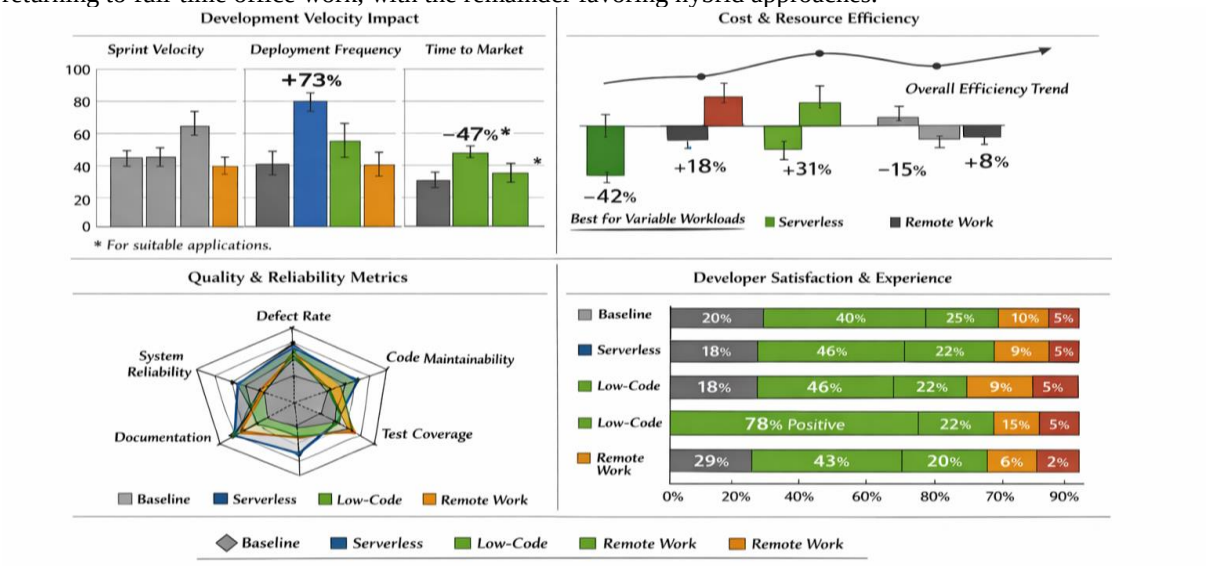

Figure 3: Comparative Performance Analysis Across Modern Practices

DOI: 10.46121/pspc.50.4.4

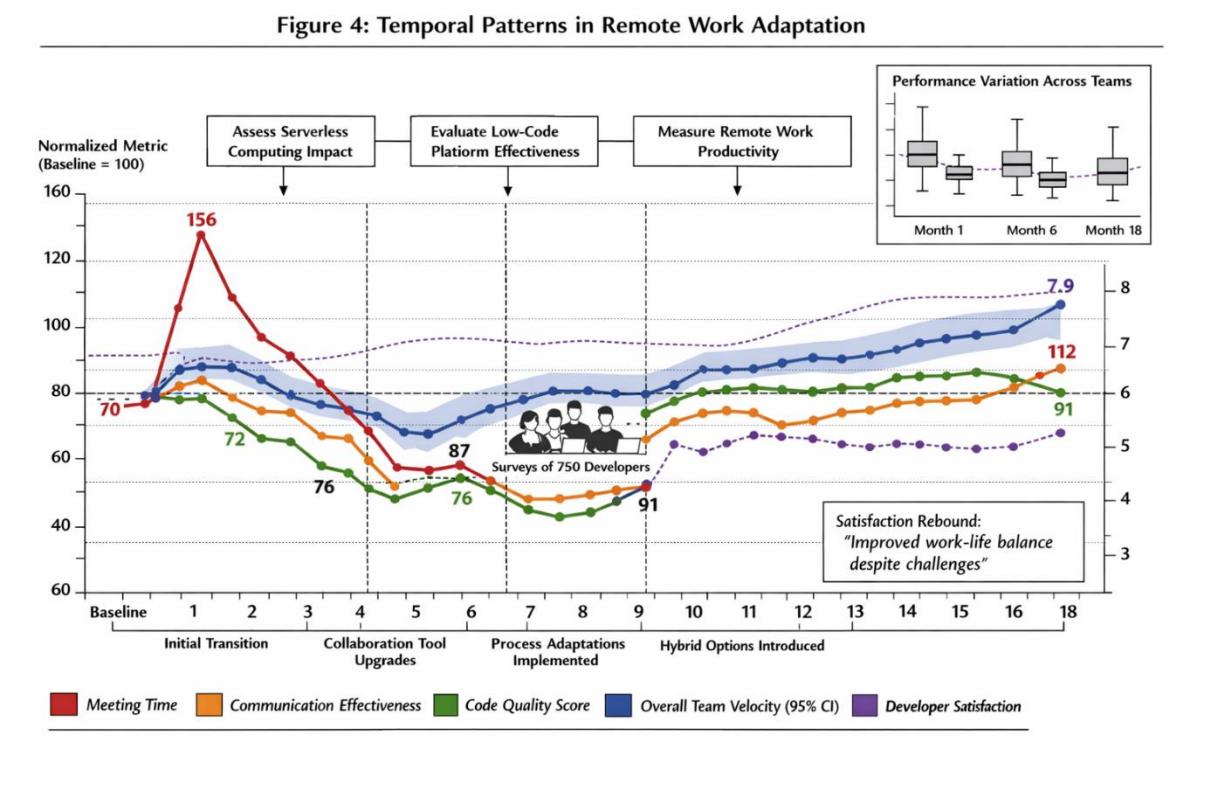

Figure 4: Temporal Patterns in Remote Work Adaptation

Table 3: Serverless Performance Metrics Summary

Metric	Traditional Baseline	Serverless Performance	Change (%)	Statistical Significance
Deployment Frequency (per month)	4.2	7.3	+73%	$p < 0.001$
Infrastructure Management Time (hours/month)	42	17.6	-58%	$p < 0.001$
Median Response Time (ms)	145	128	-12%	$p < 0.01$
95th Percentile Response Time (ms)	380	448	+18%	$p < 0.05$
Cost (variable traffic applications)	$100	$58	-42%	$p < 0.001$
Cost (consistent traffic applications)	$100	$118	+18%	$p < 0.05$
Cold Start Rate (% of invocations)	N/A	22%	N/A	N/A
Error Rate (%)	0.8	1.2	+50%	$p < 0.05$

DOI: 10.46121/pspc.50.4.4

Table 4: Development Efficiency Comparison

Application Type	Traditional Dev Time (weeks)	Low-Code Dev Time (weeks)	Time Savings (%)	Defect Rate Traditional	Defect Rate Low-Code	Quality Change
Simple Forms/CRUD	8.5	4.5	47%	12.3/1000 LOC	9.5/1000 LOC	-23% defects
Workflow Automation	6.2	3.8	39%	8.7/1000 LOC	7.1/1000 LOC	-18% defects
Dashboard/Reporting	5.8	3.5	40%	6.2/1000 LOC	5.8/1000 LOC	-6% defects
Custom Business Logic	12.4	10.9	12%	15.2/1000 LOC	16.8/1000 LOC	+11% defects
Complex Integration	14.6	15.8	-8%	18.5/1000 LOC	21.3/1000 LOC	+15% defects
Developer Satisfaction	N/A	N/A	N/A	7.2/10	6.8/10	-6%

DISCUSSION

6.1 Interpretation of Findings

The research findings reveal that modern development practices deliver significant benefits but require careful matching to appropriate contexts. Serverless computing shows clearest value for applications with variable workloads, event-driven architectures, and teams seeking to minimize operational overhead. The 58% reduction in infrastructure management time represents substantial efficiency gains, freeing developers to focus on feature development rather than operational concerns. However, the performance variability from cold starts and increased error rates indicate that serverless remains unsuitable for latency-sensitive applications requiring consistent response times.

Low-code platforms demonstrate strong value propositions for specific application categories while disappointing for others. The 47% development time reduction for simple forms and workflows confirms vendor claims about productivity benefits, but only when problem domains align with platform capabilities. The dramatic effectiveness decline for complex applications—and actual time increases for integration-heavy systems—reveals fundamental limitations in current low-code approaches. Organizations must carefully assess whether specific projects match low-code strengths before committing to these platforms.

Remote work outcomes proved more variable than either serverless or low-code results, reflecting the importance of organizational and team factors beyond mere technology deployment. The eventual 2% productivity improvement masks enormous variation, with some teams thriving remotely while others struggled persistently. Success factors included existing team cohesion, senior developer presence for mentoring, strong asynchronous communication cultures, and adequate tooling. Organizations cannot simply mandate remote work and expect productivity maintenance; they must intentionally build supporting practices and infrastructure.

6.2 Practical Implications

For organizations considering serverless adoption, the research suggests starting with new applications exhibiting suitable characteristics rather than wholesale migration of existing systems. Event-driven microservices, API backends with variable traffic, and data processing pipelines represent ideal initial use cases. Organizations should implement comprehensive monitoring to track cold start rates and establish performance budgets that account for serverless latency characteristics. The cost benefits for variable workloads justify adoption even with some performance tradeoffs, but consistent-traffic applications require careful cost analysis.
Low-code platform adoption should follow a portfolio approach, using these tools for applications matching their strengths while continuing traditional development for complex systems. Organizations should establish

DOI: 10.46121/pspc.50.4.4

clear criteria for low-code suitability, likely including maximum complexity thresholds, integration requirements, and performance expectations. Developer training on both platform capabilities and limitations helps teams make informed tool choices rather than forcing inappropriate applications into low-code molds.

Remote work strategies require recognizing that not all teams and individuals thrive equally in distributed environments. Organizations should provide choice when possible, offering hybrid options that accommodate different working styles and life circumstances. Investment in collaboration tools, asynchronous communication training, and documentation culture yields returns in productivity and quality. Regular team building and opportunities for in-person interaction help maintain cohesion despite physical distribution.

6.3 Limitations and Constraints

Several limitations constrain generalizability of these findings. The serverless evaluation focused primarily on AWS Lambda and Azure Functions, potentially missing characteristics of other serverless platforms. Application selection emphasized web services and data processing, excluding domains like gaming, IoT, or scientific computing where patterns might differ significantly.

Low-code assessment examined a relatively small project sample of 30 applications, and vendor platform selection (primarily Microsoft Power Apps and Mendix) may not represent all low-code approaches. The matching process for comparing low-code against traditional development, while rigorous, could not perfectly control for all confounding variables like team skill and organizational context.

Remote work analysis occurred during exceptional pandemic circumstances that may not reflect normal remote work conditions. The forced nature of transitions, combined with broader pandemic stressors, potentially confounded results. Organizations with existing remote work experience showed different patterns than those transitioning for the first time, but sample sizes limited ability to analyze these subgroups separately.

6.4 Future Research Directions

Several promising research directions emerge from this work. Long-term studies tracking serverless applications beyond the six-month evaluation period would reveal whether operational benefits persist and whether teams develop patterns to mitigate performance challenges. Research examining serverless adoption for emerging use cases like machine learning inference and edge computing could expand understanding of appropriate contexts.

Low-code platform research should examine the next generation of tools incorporating AI-assisted development and more sophisticated customization capabilities. Longitudinal studies tracking low-code application maintenance over multi-year periods would clarify whether long-term costs offset initial development savings. Research comparing different low-code vendors' approaches and capabilities would help organizations select appropriate platforms.

Remote work research should examine hybrid models combining remote and office work, as many organizations adopt these approaches long-term. Studies investigating specific practices that distinguish successful from struggling remote teams would provide actionable guidance. Research examining remote work impacts on innovation, knowledge creation, and organizational culture would address strategic questions beyond immediate productivity.

CONCLUSION

This research provides empirical evidence about modern development practices that are reshaping software engineering. Through systematic evaluation of serverless computing, low-code platforms, and remote work environments, we have identified both substantial benefits and important limitations that organizations must understand for successful adoption.

DOI: 10.46121/pspc.50.4.4

Serverless computing delivers meaningful operational efficiency improvements, reducing infrastructure management overhead by 58% and increasing deployment frequency by 73%. However, performance variability from cold starts and cost structures that favor variable workloads mean that serverless suitability depends critically on application characteristics. Organizations should adopt serverless strategically for appropriate use cases rather than pursuing wholesale architectural transformation.

Low-code platforms accelerate development by 40-50% for applications matching their design assumptions—primarily simple forms, workflows, and dashboards. Yet effectiveness declines sharply as complexity increases, with custom business logic showing minimal benefits and integration-heavy applications actually taking longer to develop. The research confirms that low-code represents a valuable tool for specific scenarios rather than a universal development approach. Organizations should maintain portfolio strategies using low-code where it excels while continuing traditional development for complex systems.

Remote work maintains or improves productivity for well-supported teams with appropriate characteristics, but creates persistent challenges for others. The 18-month adaptation period revealed initial disruption followed by recovery and eventual modest productivity improvements. However, aggregate statistics mask enormous variation, with success depending on team composition, organizational culture, and intentional investment in remote-enabling practices. Organizations must recognize that remote work requires active cultivation rather than passive acceptance of distributed teams.

Looking forward, modern development practices will likely continue evolving as platforms mature and organizations gain experience. Serverless architectures may address current performance limitations through improved cold start optimization and enhanced debugging tools. Low-code platforms incorporating AI assistance may expand their applicable scope beyond current constraints. Remote work practices will likely stabilize into hybrid models balancing distributed flexibility with periodic in-person collaboration.

Organizations navigating these transformations should adopt experimental mindsets, carefully measuring impacts rather than assuming benefits. The diversity of outcomes across teams and contexts in this research emphasizes that successful adoption requires matching practices to specific organizational circumstances. Modern development practices offer genuine improvements over traditional approaches, but realizing their potential demands thoughtful implementation informed by empirical evidence about what works, for whom, and under what conditions.

REFERENCES

1. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A. and Suter, P. (2017) 'Serverless computing: Current trends and open problems', in *Research Advances in Cloud Computing*, Singapore: Springer, pp. 1-20.
2. Bellomo, S., Ernst, N., Nord, R. and Kazman, R. (2021) 'Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail', *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, pp. 290-299.
3. Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L. and Pallickara, S. (2018) 'Serverless computing: An investigation of factors influencing microservice performance', *Proceedings of IEEE International Conference on Cloud Engineering*, pp. 159-169.
4. Miller, C., Rodeghero, P., Storey, M., Ford, D. and Zimmermann, T. (2021) 'How was your weekend? Software development teams working from home during COVID-19', *Proceedings of the 43rd International Conference on Software Engineering*, pp. 624-636.

DOI: 10.46121/pspc.50.4.4

5. Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., Zhou, M., Turhan, B., Hoda, R., Hata, H., Robles, G., Milani Fard, A. and Alkadhi, R. (2020) 'Pandemic programming: How COVID-19 affects software developers and how their organizations can help', *Empirical Software Engineering*, 25(6), pp. 4927-4961.
6. Roberts, M. (2018) 'Serverless architectures', *Martin Fowler's Blog*, Available at: https://martinfowler.com/articles/serverless.html (Accessed: 15 March 2022).
7. Russo, D., Hanel, P.H., Altnickel, S. and van Berkel, N. (2021) 'Predictors of well-being and productivity among software professionals during the COVID-19 pandemic: A longitudinal study', *Empirical Software Engineering*, 26(6), pp. 1-63.
8. Sahay, A., Indamutsa, A., Di Ruscio, D. and Pierantonio, A. (2020) 'Supporting the understanding and comparison of low-code development platforms', *Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 171-178.
9. Waszkowski, R. (2019) 'Low-code platform for automating business processes in manufacturing', *IFAC-PapersOnLine*, 52(10), pp. 376-381.

DOI: 10.46121/pspc.50.4.4