Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 3, Issue 3

Open Access | PP: 01-07

DOI: https://doi.org/10.70315/uloap.ulirs.2026.0303001

Frameworks for Transitioning from Technology-Siloed to Cross-Functional Engineering Teams in Modern Software Organizations

Michael Rainesh

Director of Engineering, Portside, Inc., East Gwillimbury, Canada.

Abstract

Software organizations often keep frontend, backend, mobile, infrastructure, and quality-assurance engineers in separate units long after product delivery has begun to depend on joint ownership across these domains. The separation protects specialist knowledge, yet it creates delays when one feature moves through several queues before release. This review examines phased transition patterns from technology-siloed structures toward cross-functional engineering teams. It draws on recent software engineering literature on agile teamwork, DevOps structures, continuous delivery, platform teams, scaled autonomy, communities of practice, and human factors in agile projects. The method combines comparative source analysis, conceptual synthesis, typological classification, and analytical generalization. The review identifies structural limits of silo-based delivery, compares transition mechanisms, and develops a practical interpretation of staged ownership transfer. The proposed approach links cross-functional restructuring with federated governance, enabling teams, quality practices embedded in development, and monitoring indicators covering flow, reliability, collaboration, capability, and sustainability.

Keywords: *Cross-Functional Teams, Software Engineering, Agile Transformation, DevOps, Platform Teams, Federated Governance, Team Autonomy, Continuous Delivery, Engineering Management, SDLC.*

INTRODUCTION

Many software organizations grow around technical specialization. Frontend developers work in one group, backend developers in another, mobile engineers in a third. Infrastructure and quality-assurance specialists often sit outside product delivery teams. This arrangement offers clear hiring profiles, coherent mentoring paths, and a natural place for deep technical knowledge. It also creates a delivery pattern in which one customer-facing feature moves through several handoffs before users receive it.

Product work in current engineering organizations rarely fits one technical layer. A feature can touch user interface logic, service contracts, data permissions, API behavior, release automation, and post-release monitoring. A siloed structure divides this work before the team understands its full delivery path. Managers then spend time coordinating dependencies that the team structure itself has created. Engineers wait for another group to finish a related task, product owners negotiate priority across several backlogs, and quality work arrives late in the cycle.

The shift toward cloud-native delivery, DevOps practices, continuous integration, continuous delivery, and product-oriented accountability has made these handoffs more visible. Short release cycles leave little room for sequential queues. Teams that design, build, test, deploy, and monitor their product area can make smaller decisions faster and can learn from production feedback with fewer intermediaries. Cross-functional engineering teams emerged from this pressure, although their adoption often suffers when organizations copy the label without changing ownership, governance, and engineering habits.

The aim of this article is to review and synthesize phased approaches for moving from technology-siloed engineering structures to cross-functional, cross-technology teams in modern software organizations. The article treats transition as a review-based organizational design problem, not as a validated proprietary framework.

Three research objectives structure the analysis. The first objective identifies structural limits of technology-siloed engineering models in organizations that rely on agile,

Citation: Michael Rainesh, "Frameworks for Transitioning from Technology-Siloed to Cross-Functional Engineering Teams in Modern Software Organizations", Universal Library of Innovative Research and Studies, 2026; 3(3): 01-07. DOI: https://doi.org/10.70315/uloap.ulirs.2026.0303001.

DevOps, and continuous delivery practices. The second objective compares transition mechanisms discussed in recent literature, including product teams, platform teams, communities of practice, scaled autonomy, and governance through enabling units. The third objective develops an implementation-oriented interpretation of how engineering leaders can monitor phased transition through delivery, quality, collaboration, and capability indicators while keeping practitioner-reported improvements separate from controlled empirical findings.

The contribution of the article lies in connecting organizational design, DevOps team structures, agile-team effectiveness, and governance mechanisms. Cross-functional restructuring often appears in managerial discourse as a target state. This review treats it as a staged transfer of ownership, knowledge, quality responsibility, and decision rights. The guiding hypothesis states that transition from technology silos to cross-functional engineering teams improves organizational responsiveness only when teams receive autonomy together with shared engineering standards, platform support, and explicit quality controls.

MATERIALS AND METHODS

The review draws on a focused literature search conducted across Scopus-indexed and publisher-hosted repositories, including ScienceDirect, SpringerLink, ACM Digital Library, IEEE Computer Society, and selected industry repositories whose practitioner models have shaped software-engineering discourse. Search strings combined "cross-functional teams," "software engineering teams," "DevOps team structures," "continuous delivery," "platform teams," "agile teamwork effectiveness," "scaled autonomy," "communities of practice," "Team Topologies," and "Spotify model." The selection primarily retained works published between 2021 and 2025 that addressed engineering-team organization, agile or DevOps delivery, continuous delivery, platform support, scaled autonomy, or knowledge-sharing structures. One earlier practitioner source was retained as a primary reference for the Team Topologies model because the article uses it as an industry vocabulary rather than as empirical evidence. The screening excluded generic project-management papers, non-software product development studies, promotional consulting material without analytical value, and empirical papers too narrow for organizational synthesis. The search produced 46 records. Title and abstract screening removed 22 records. Full-text or extended abstract assessment removed another 14. The final corpus contained nine recent academic sources and one practitioner primary source.

The methods combine comparative source analysis, source-based conceptual synthesis, typologization, and analytical generalization. Comparative analysis supports the first objective by contrasting siloed departments, classical DevOps arrangements, cross-functional teams, and platform-supported structures. Source analysis supports the second objective by tracing how the literature treats autonomy, shared ownership, communities of practice, and enabling teams. Conceptual synthesis supports the third objective by converting these materials into a phased transition logic with monitoring categories related to flow, quality, collaboration, capability, governance, and sustainability.

RESULTS

Recent software-engineering literature treats team structure as part of the delivery system. It does not appear as a neutral administrative background. The clearest illustration comes from research on continuous delivery and DevOps organization. A grounded-theory study of software teams pursuing continuous delivery distinguishes siloed departments, classical DevOps arrangements, cross-functional teams, and platform teams as recurring structures (Leite et al., 2021). This taxonomy separates tool modernization from organizational modernization. A company can automate builds, move workloads to cloud infrastructure, and introduce deployment pipelines while work still travels through separate technical queues. For the first objective of this article, this distinction explains why technology silos keep shaping delivery even after organizations adopt modern toolchains.

Technology-siloed structures protect expertise by grouping similar specialists together. The cost appears when one feature crosses several technical domains. Continuous-delivery research describes how organizations divide work between development and infrastructure professionals and how those divisions shape release preparation (Leite et al., 2021). A frontend-backend-mobile split produces similar friction in customer-facing systems. Users experience a feature as one product change. The organization plans it as several technical fragments. Each fragment can carry its own backlog, review cycle, and release dependency. The larger the chain, the more coordination labor falls on managers, architects, release leads, or informal technical brokers.

Research on DevOps team structures adds another layer. An exploratory study using interviews, workshops, and observations in multinational software-intensive companies classifies DevOps arrangements through variables such as collaboration frequency, shared product ownership, autonomy, and support from horizontal teams (López-Fernández et al., 2022). The study links delivery structure with ownership and support mechanisms instead of treating DevOps as a toolbox. Its relevance for cross-functional transition is direct. Moving several specialists into one reporting unit will not create cross-functionality by itself. Teams need authority over a product area, routines for joint decision-making, and access to enabling groups that help them use shared engineering practices.

Agile teamwork research clarifies why some structural

changes produce durable improvement while others remain cosmetic. A teamwork effectiveness model for agile software development identifies shared leadership, team orientation, redundancy, adaptability, and peer feedback as major contributors to team performance (Strode et al., 2022). These factors recast transition as a capability-building process. A team can carry the title "full-stack" while each engineer still owns one narrow layer. In that case, mobile work still waits for the mobile specialist, backend changes still wait for the backend specialist, and deployment knowledge still sits with one person. Cross-functional work begins when engineers develop enough overlapping knowledge to review, pair, diagnose, and make product decisions together.

A broader systematic review of agile-team effectiveness organizes performance through input factors, mediating processes, team states, and outcomes (Steegh et al., 2025). This model helps explain why structure alone rarely changes delivery performance at the pace leaders expect. Team design creates conditions for improved work, but daily behavior decides whether those conditions become capability. Teams need feedback routines, coordination norms, learning habits, and leadership behavior that reduces dependence on former silos. A team renamed as cross-functional will still work as a set of specialists if members lack time and support to learn adjacent technologies.

Autonomy requires more discipline than many transformation programs admit. Research on decentralized decision-making at Spotify reports that scaled autonomy operates through decentralized authority, workgroups, managerial compromises, and alignment mechanisms across teams (Šmite et al., 2023). The lesson is relevant for organizations tempted to copy squad, tribe, chapter, or guild labels. The vocabulary travels faster than the operating discipline behind it. Teams need room to solve product problems, yet they also need shared standards, architecture guardrails, and cross-team forums that keep local choices from becoming long-term fragmentation.

Communities of practice offer one path through this tension. Research on communities of practice in large-scale agile development reports that organizations use them for knowledge exchange, alignment, and adoption of agile practices across teams (Tobisch et al., 2024). This finding helps bridge technology silos and cross-functional teams. If a company distributes backend engineers across product teams, backend expertise should not disappear from the organization. It can move into a chapter, guild, community, or Center of Excellence where engineers maintain standards, mentor colleagues, and discuss technical direction. Product teams gain delivery responsibility, while specialist knowledge remains visible and teachable.

Human factors research introduces a constraint that restructuring plans often miss. A study of agile software development projects identifies team capability and customer involvement as strong success factors, while psychological safety and autonomy shape success through more complex relationships (Barros et al., 2024). Cross-functional transition exposes engineers to unfamiliar code, shared accountability, and closer contact with product consequences. If team members fear blame for mistakes in adjacent domains, ownership transfer stalls. Developers avoid unfamiliar services. Testers keep acting as late-stage gatekeepers. Product owners continue to route work toward former specialists. Trust and competence have to grow together.

Technical integration research supports the same conclusion from another angle. A systematic literature review on Agile, Cloud, and DevOps integration reports benefits related to collaboration and faster development, while challenges involve tool proliferation, testing, mindset, and integration complexity (El Aouni et al., 2024). Cross-functional teams need more than a revised team chart. Engineers require shared pipelines, reliable test automation, environment self-service, deployment visibility, and feedback from production systems. Without these foundations, leaders expand team responsibility while leaving teams without the means to exercise that responsibility.

Research on DevOps performance gives a further reason to reject universal templates. Work on context-capability coalignment in DevOps teams argues that performance depends on fit between organizational setting and DevOps capabilities, with coalignment treated as a continuing process (Plant et al., 2025). A regulated enterprise, a small SaaS provider, a global product company, and a department maintaining legacy systems face different constraints. Release risk, architecture coupling, compliance demands, and team maturity shape the safe pace of ownership transfer. A phased framework respects these differences better than a single reorganization model.

The Team Topologies practitioner model has shaped industry discussion through its focus on organizational design, software architecture, cognitive load, and team interaction patterns (Skelton & Pais, 2019). For a review-based article, its value lies in vocabulary rather than empirical proof. Stream-aligned teams, platform teams, enabling teams, and complicated-subsystem teams offer a way to discuss who owns product value, who reduces cognitive load, who transfers practices, and which domains require specialist stewardship. This language complements empirical work on continuous delivery and DevOps structures because it treats team boundaries and interaction modes as engineering design choices. Figure 1 presents the transition logic derived from the reviewed sources. The sequence reflects organizational categories in continuous-delivery research, DevOps team-structure research, and Team Topologies thinking. It does not add numerical claims.

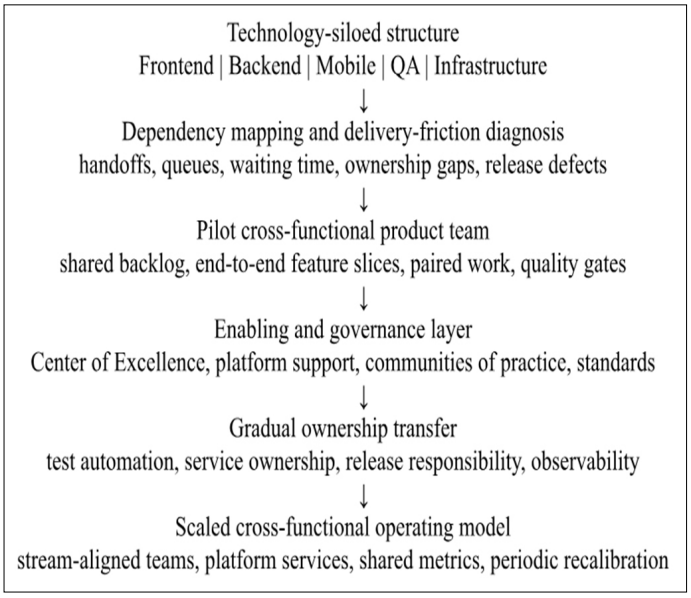

Figure 1. Phased transition logic from technology silos to cross-functional engineering teams (adapted from Leite et al., 2021, López-Fernández et al., 2022, and Skelton & Pais, 2019)

The figure frames transition as a staged movement. It begins with dependency diagnosis. Engineering leaders need to see where work waits, where knowledge concentrates, where release quality depends on a separate gatekeeping group, and where managers compensate for structural friction through manual coordination. A pilot cross-functional team has value only after the organization has identified these delivery constraints.

The first comparison across sources concerns ownership. Continuous-delivery research shows that organizations divide work between development and infrastructure professionals in different ways (Leite et al., 2021). DevOps team-structure research links consolidated team models with shared product ownership and autonomy (López-Fernández et al., 2022). Agile-team effectiveness research adds shared leadership, peer feedback, and adaptability as mechanisms inside the team (Strode et al., 2022). These sources point to one operational conclusion. Ownership transfer requires formal authority and daily routines. A team needs control over a product slice, while engineers need habits that let them review work across domains, pair on unfamiliar changes, and accept joint responsibility for quality.

The second comparison concerns autonomy. Spotify-related research presents scaled autonomy as a managed condition supported by decentralized authority and alignment mechanisms (Šmite et al., 2023). Research on agile project success links autonomy with capability and psychological safety (Barros et al., 2024). DevOps coalignment research connects team performance with organizational setting and capability maturity (Plant et al., 2025). These findings support phased autonomy. Early autonomy can cover backlog execution and local technical choices. Wider autonomy over release, architecture, and operational response should follow when a team has platform support, reliable tests, and production observability.

The third comparison concerns knowledge continuity. Communities of practice research shows that large-scale agile organizations use cross-team forums to maintain learning and alignment (Tobisch et al., 2024). Agile-team effectiveness literature links performance with mediating processes and team states, which means knowledge moves through interaction as much as documentation (Steegh et al., 2025). Team Topologies thinking treats interaction modes and boundaries as design choices that affect cognitive load (Skelton & Pais, 2019). These sources support a practical rule: an organization should not dissolve silos and leave their knowledge function unattended. Backend chapters, QA enablement groups, platform teams, and Centers of Excellence can preserve standards while product teams take greater delivery responsibility.

The fourth comparison concerns quality. Agile-Cloud-DevOps integration literature identifies testing, tool complexity, mindset, and automation as conditions for successful integration (El Aouni et al., 2024). DevOps team-structure research shows that platform teams and Centers of Excellence can support product teams through mentoring and shared technical capability (López-Fernández et al., 2022). Human-factor research links agile success with team capability and psychological safety (Barros et al., 2024). These positions suggest that a QA-less or reduced-QA model should be described as relocation of quality work into engineering practice. Developers take responsibility through test automation, code review, pairing, observability, and release-readiness routines. Removing a QA gate without building these practices increases risk.

Enterprise practitioner experience can illustrate this point without turning it into empirical evidence. In large organizations, federated governance through a Center of Excellence can let business units keep independent development teams while shared standards, release expectations, and technical guidance remain visible. Pair programming and gradual ownership transfer align with the literature because they reduce knowledge concentration and distribute quality responsibility. Practitioner-reported improvements such as higher throughput, faster delivery, or fewer production release issues should appear as field experience, not as controlled research findings. Their value lies in showing how literature-informed principles can enter operational leadership, especially during movement from QA-dependent delivery toward developer-owned quality practices.

The synthesis yields five premises for transition. Technology silos solve specialization problems while creating dependency costs in feature delivery. Cross-functional teams require end-to-end product ownership, not only a mixed skill list.

Technical depth needs preservation through platform teams, communities of practice, chapters, guilds, or Centers of Excellence. Autonomy needs governance so that teams do not drift into incompatible tools and release standards. Quality has to become an engineering system built from automated tests, peer review, observability, release discipline, and learning loops.

DISCUSSION

Cross-functional transformation works best when engineering leaders treat it as organizational refactoring. In code, refactoring improves structure while preserving behavior. In an engineering department, the comparable task is to improve delivery structure while preserving expertise, reliability, and cultural continuity. A rushed conversion of frontend, backend, mobile, QA, and infrastructure groups into product teams can reduce formal handoffs while creating hidden fragility. Engineers may keep old specialist identities inside the new structure. Quality work may lose its owner. Architecture decisions may diverge. Communities of practice may weaken if managers celebrate autonomy but fail to fund shared learning.

A phased implementation model reduces these risks. The first phase maps delivery dependencies and identifies where work waits. Teams inspect feature flow, release preparation, defect leakage, review bottlenecks, environment delays, and repeated escalation points. The second phase creates a pilot cross-functional team around a bounded product slice. The pilot receives a shared backlog, release responsibility, paired work across technologies, and a compact set of quality standards. The third phase builds enabling structures. A platform group, Center of Excellence, or technical guild supplies reusable pipelines, testing templates, observability standards, and architecture guardrails. The fourth phase transfers ownership in steps. Developers expand code-review participation, then test automation responsibility, then deployment readiness, then production observability. Scaling begins only after the pilot demonstrates reduced friction with stable quality.

Decision logic matters because different work types need different team shapes. A customer-facing product stream benefits from end-to-end ownership because feature flow crosses several layers. A specialized algorithmic subsystem may need a team built around rare expertise. A deployment platform needs product-style management because internal engineers rely on it every day. A regulatory or security function may operate through governance and enablement. Leaders should select the target structure from work type, coupling, release frequency, risk profile, and learning capacity. Table 1 compares organizational structures that often appear during transition. Its purpose is to support fit-for-purpose reasoning. The table compares each structure by organizing principle, strength, risk, and transition use.

Table 1. Comparative logic of engineering-team structures in transition from technology silos to cross-functional delivery (compiled by the author based on Leite et al., 2021, López-Fernández et al., 2022, Šmite et al., 2023, and Skelton & Pais, 2019)

Structure	Main organizing principle	Typical strength	Typical risk	Suitable transition use
Technology-siloed departments	Engineers grouped by technical domain	Deep specialization and clear skill ownership	Handoffs, queues, fragmented feature responsibility	Baseline structure for dependency diagnosis
Classical DevOps arrangement	Development and operations linked through collaboration routines	Better release coordination and shared operational awareness	Persistent boundary between build and run responsibilities	Intermediate step where infrastructure ownership cannot move at once
Cross-functional product team	Team owns a product slice across technologies	Faster end-to-end feature flow and clearer accountability	Skill gaps, uneven standards, hidden dependence on former specialists	Main target for customer-facing product streams
Platform-supported model	Product teams consume internal platform capabilities	Lower cognitive load and standardized delivery paths	Platform backlog becomes a new bottleneck if leaders treat it as a service desk	Scaling mechanism after pilot teams prove ownership readiness
Federated governance with CoE	Teams retain local autonomy under shared standards	Balance between independence and organizational consistency	Governance becomes symbolic if standards lack enforcement and coaching	Enterprise setting with several business units or product lines

The comparison indicates that the strongest transition design relocates specialization instead of erasing it. Specialist knowledge moves from delivery queues into enabling systems, reusable platforms, coaching structures, and decision forums. Cross-functional delivery does not require every engineer to reach the same level in every technology. It requires enough shared capability inside the team to move product work without routine external handoffs, plus enough support outside the team to learn without damaging quality.

Monitoring should follow the same logic. A dashboard focused only on velocity encourages weak behavior. Throughput can rise while defect leakage grows. Release issues can fall while engineers absorb unsustainable pressure. Collaboration can look active in meetings while actual knowledge remains concentrated in one or two people. Metrics should cover flow, quality, collaboration, capability, governance, and sustainability. Table 2 proposes a monitoring structure for phased transition. It supports review-based application and does not assume controlled before-after data.

Table 2. Monitoring indicators for phased transition to cross-functional engineering teams (compiled by the author based on Strode et al., 2022, Barros et al., 2024, Steegh et al., 2025, El Aouni et al., 2024, and Plant et al., 2025)

Monitoring dimension	Indicator examples	Interpretation during transition	Warning sign
Flow	Lead time, cycle time, blocked-work ratio, dependency wait time	Shows whether cross-functional structure reduces coordination delays	Faster starts but unchanged release completion
Quality	Escaped defects, release rollback frequency, test coverage trend, review defect density	Shows whether quality responsibility has moved into engineering practice	QA removal with no compensating automation or peer-review discipline
Collaboration	Pairing frequency, cross-domain reviews, shared incident participation, backlog refinement participation	Shows whether work crosses skill boundaries inside the team	One specialist still handles each layer alone
Capability	Skill matrix movement, onboarding time, platform adoption, practice reuse	Shows whether the team gains durable cross-technology competence	Delivery depends on undocumented individual knowledge
Governance	Standard compliance, architecture exceptions, platform support response, CoE coaching activity	Shows whether autonomy remains aligned with organizational standards	Each team invents incompatible tools and release rules
Sustainability	Workload balance, interruption rate, after-hours incident load, retention signals	Shows whether broader ownership remains humane and stable	Higher output paired with fatigue and silent quality debt

This monitoring structure separates transformation success from a single efficiency figure. Practitioner-reported gains such as 20 percent throughput growth, 80 percent reduction in production release issues, or 40 percent faster delivery can help managers notice operational movement. Academic writing should frame such figures as experience-based indicators unless a source provides documented baselines, protocols, and reproducible measurement procedures. Without those materials, the figures illustrate possible operational effects. They do not prove causal impact.

A workable implementation sequence begins with one product stream whose architecture has enough modularity and whose leadership can protect the team from constant interruption. The pilot team needs an ownership charter. The charter defines which services, interfaces, tests, deployment steps, and operational signals belong to the team. Pair programming fits best where knowledge has to move from a former specialist to wider team membership. Code review should train judgment, not only block defects. QA specialists, where they remain in the structure, should move toward test strategy, automation design, risk analysis, and coaching. Teams without dedicated QA need stronger checklists, automated regression suites, release-readiness criteria, and defect-review routines.

Federated governance suits enterprise organizations because it avoids weak extremes. Full centralization slows teams and recreates bottlenecks. Unbounded autonomy leads to inconsistent engineering practices. A Center of Excellence or comparable enabling unit defines minimum standards, maintains reusable templates, supports platform adoption, and intervenes when teams lack capability. Its authority should remain practical. It coaches, measures, and improves standards with teams instead of acting as a remote policy office.

The authorial position in this review is direct. Cross-functional transformation should be judged by movement of responsibility, not by team labels. A team has transitioned when feature work no longer waits for external layers as a routine pattern, quality practices appear throughout the SDLC, production feedback enters planning, and technical standards remain coherent across teams. Maturity becomes visible when a team can absorb moderate product change without repeated managerial escalation, emergency coordination, or hidden dependence on a former silo.

CONCLUSION

Technology-siloed engineering structures preserve specialist knowledge while creating dependency costs in product delivery. The reviewed literature links these costs with queues, fragmented responsibility, release coordination work, and reliance on boundary-spanning managers or informal experts. Transition begins with diagnosis of those dependencies before leaders redesign the team structure.

Recent studies and practitioner models converge on a staged solution. Product-oriented teams need autonomy, shared ownership, peer feedback, and adaptability. At scale, organizations need platform teams, communities of practice, chapters, guilds, or Centers of Excellence to preserve technical standards while product teams assume delivery responsibility. The most defensible model combines federated engineering governance with visible standards, active enablement, and gradual ownership transfer.

The hypothesis receives support from the synthesis. Cross-functional teams improve responsiveness when autonomy grows together with engineering capability, quality safeguards, shared standards, and platform support. Practitioner-reported improvements in throughput, delivery speed, and release quality can serve as field signals, but they should not replace controlled evidence. The practical contribution lies in the transition logic: map dependencies, pilot cross-functional ownership, build enabling structures, transfer quality responsibility step by step, monitor several dimensions at once, and scale only after teams demonstrate reliable end-to-end delivery.

REFERENCES

1. Barros, L., Tam, C., & Varajão, J. (2024). Agile software development projects–unveiling the human-related critical success factors. *Information and Software Technology, 170*, 107432. https://doi.org/10.1016/j.infsof.2024.107432
2. El Aouni, F., Moumane, K., Idri, A., Najib, M., & Jan, S. U. (2024). *Systematic literature review on agile, cloud, and DevOps integration: Challenges, benefits*. https://doi.org/10.2139/ssrn.4827875
3. Leite, L., Pinto, G., Kon, F., & Meirelles, P. (2021). The organization of software teams in the quest for continuous delivery: A grounded theory approach. *Information and Software Technology, 139*, 106672. https://doi.org/10.1016/j.infsof.2021.106672
4. Lopez-Fernandez, D., Diaz, J., Garcia, J., Perez, J., & Gonzalez-Prieto, A. (2022). DevOps team structures: Characterization and implications. *IEEE Transactions on Software Engineering, 48*(10), 3716–3736. https://doi.org/10.1109/TSE.2021.3102982
5. Plant, O. H., Aldea, A., & Van Hillegersberg, J. (2025). Improving DevOps team performance through context-capability coalignment: Towards a profile for public sector organizations. *Information and Software Technology, 178*, 107585. https://doi.org/10.1016/j.infsof.2024.107585
6. Skelton, M., & Pais, M. (2019). *Team topologies: Organizing business and technology teams for fast flow*. IT Revolution Press. https://teamtopologies.com/book
7. Šmite, D., Moe, N. B., Floryan, M., Gonzalez-Huerta, J., Dorner, M., & Sablis, A. (2023). Decentralized decision-making and scaled autonomy at Spotify. *Journal of Systems and Software, 200*, 111649. https://doi.org/10.1016/j.jss.2023.111649
8. Steegh, R., Van De Voorde, K., & Paauwe, J. (2025). Understanding how agile teams reach effectiveness: A systematic literature review to take stock and look forward. *Human Resource Management Review, 35*(1), 101056. https://doi.org/10.1016/j.hrmr.2024.101056
9. Strode, D., Dingsøyr, T., & Lindsjorn, Y. (2022). A teamwork effectiveness model for agile software development. *Empirical Software Engineering, 27*(2), 56. https://doi.org/10.1007/s10664-021-10115-0
10. Tobisch, F., Schmidt, J., & Matthes, F. (2024). Investigating communities of practice in large-scale agile software development: An interview study. In D. Šmite, E. Guerra, X. Wang, M. Marchesi, & P. Gregory (Eds.), *Agile processes in software engineering and extreme programming* (Vol. 512, pp. 3–19). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-61154-4_1