INTERNATIONAL JOURNAL OF BUSINESS AND MANAGEMENT SCIENCES (ISSN: 2693-3500)
VOL.06 ISSUE 10 (2026)

*Research Article*

# Inter-Team Interfaces as the Primary Source of Losses in Large-Scale Projects

**Nikolaev Denis**
Project manager, hh.ru Dubai, UAE

Received: 04 September 2026
Revised: 18 September 2026
Accepted: 24 September 2026
Published: 09 October 2026

Doi: 10.55640/ijbms-06-10-02
Page No: 10-19

**Copyright:** © 2026 Authors retain the copyright of their manuscripts, and all Open Access articles are disseminated under the terms of the Creative Commons Attribution License 4.0 (CC-BY), which licenses unrestricted use, distribution, and reproduction in any medium, provided that the original work is appropriately cited.

**Abstract**

Large-scale software projects routinely involve multiple autonomous teams whose internal processes are well-established yet whose joint output repeatedly falls short of schedule and cost targets. This paper argues that the dominant source of those losses is not within-team inefficiency but the coordination overhead generated at the points where one team's decisions must be accepted and acted upon by another. Drawing on a dual-case study from hh.ru (a major Russian labor-market platform employing more than 2,000 staff), the study compares two product launches of comparable technical scope but different inter-team topology: a three-team voice-call feature completed in six months and ranked among the company's top 18 projects of the year, and a five-team, and two senior leaders (hereafter Direction Lead A and Direction Lead B) each responsible for a distinct but adjacent major organizational direction, with no formal precedence of one over the other contact-center program characterized by substantially higher coordination friction. The analysis introduces a three-level typology of interfaces (horizontal, vertical, and cross-perimeter), identifies the non-linear growth of coordination channels consistent with Brooks's Law, and proposes the Interface Design Protocol (IDP) as a pre-launch discipline for explicitly mapping and assigning ownership of every inter-team boundary. Findings indicate that interface topology, rather than team count alone, is the decisive predictor of coordination losses. The results are relevant to project managers, program directors, and HR practitioners responsible for team structure in technology organizations.

**Keywords:** Inter-team coordination, coordination interfaces, large-scale agile, project management, multiteam systems, boundary spanning, coordination overhead, Brooks's Law, responsibility assignment.

## I. INTRODUCTION

Modern technology organizations increasingly deliver value through programs in which several autonomous product teams work in parallel on a shared objective. The Project Management Institute (PMI) reports that only 31% of projects are completed on time, within budget, and at the intended scope, while an average of 11.4% of project investment is lost to poor project performance [1]. Separately, research by McKinsey indicates that approximately 40% of large IT programs exceed their original budget, with schedule overruns averaging 7 months for projects lasting 12 to 24 months [2]. These figures have not improved substantially over the past decade despite widespread adoption of agile methods, suggesting that the root causes lie not in team-internal processes but in structural features of how teams relate to one another.

A growing body of empirical research pinpoints inter-team coordination as one of the most persistent challenges in large-scale software development [3, 4, 5]. Berntzen et al. [3] documented 27 coordination mechanisms in a single large-scale agile program and showed that the greater the number of teams, the higher the coordination demands. Dingsøyr et al. [6] found in a 12-team program that inter-team coordination was one of three dominant managerial challenges, alongside customer involvement and software architecture. Smite et al. [7] observed, in a study of Spotify's

multi-squad model, that scaled autonomy does not eliminate coordination need but displaces it to the interfaces between squads. Yet despite this accumulating evidence, project management practice has not yet produced a systematic framework for designing those interfaces before a program begins. The management of inter-team boundaries is still typically treated as an emergent, informal activity rather than a deliberate engineering discipline [3, 5].

The present paper addresses this gap through a practitioner-researcher perspective informed by direct experience managing two product programs at hh.ru. The simpler program (voice calls feature, three teams) was delivered in six months and entered the company's top-18 project shortlist from approximately 200 annual nominations. The more complex program (contact-center, five teams plus two leaders with distinct technical authority zones) exhibited markedly higher coordination friction. The contrast between these two cases, combined with a systematic review of the academic literature, yields the paper's central argument: in a multi-team project, the dominant source of schedule and resource losses is not internal team performance but the overhead concentrated at the interfaces between teams.

This paper offers three contributions. First, a three-level typology of inter-team interfaces (horizontal, vertical, and cross-perimeter) grounded in empirical evidence. Second, an analysis of how interface count and type interact non-linearly with project outcomes. Third, a practical protocol, the Interface Design Protocol (IDP), for mapping and assigning ownership of interfaces before a program launch. The scientific novelty lies in the first empirically grounded typology that distinguishes cross-perimeter interfaces (those connecting leaders with overlapping but formally distinct authority zones) as a qualitatively different and more costly category than horizontal or vertical interfaces.

The study carries certain limitations inherent in its design. The primary empirical material derives from two cases within a single company, which constrains statistical generalizability. The retrospective nature of the case analysis introduces recall bias. Metrics such as velocity loss attributable specifically to interface overhead were not formally tracked during either program and are estimated from available iteration data and retrospective accounts. These limitations are disclosed not to diminish the study's contribution but to signal the directions for follow-on validation work. Notwithstanding these constraints, the findings are practically actionable for any organization operating product teams at scale, with particular relevance to technology-intensive sectors in the United States and other knowledge-economy contexts where team-based delivery is standard practice.

## Materials and Methods

This study applies a comparative embedded case-study design in the tradition of Yin [8], supplemented by a systematic review of the peer-reviewed literature on inter-team coordination. The case-study approach is appropriate because the research question concerns a contemporary phenomenon (interface topology as a predictor of project losses) in a real organizational context where the investigator has direct access to practitioners and documents [8].

A systematic search of Scopus, IEEE Xplore, ACM Digital Library, and Springer Link was conducted using the query strings: ("inter-team coordination" OR "coordination mechanism*") AND ("large-scale" OR "multiteam" OR "project management"). This produced an initial set of 218 records. After title and abstract screening, 47 papers were read in full, and 21 were selected as primary sources on the grounds of empirical grounding, methodological rigor, and topical relevance. Analytical reports from PMI (Pulse of the Profession 2024) and the Standish Group (CHAOS 2023) were used for industry statistics, constituting less than 10% of the source base, consistent with the stated source quality criteria.

The selected literature falls into three clusters. The first cluster covers coordination mechanisms and taxonomies in large-scale agile software development [3, 4, 5, 6, 7, 9, 10]. The second cluster covers team boundary management, boundary spanning, and multiteam systems from an organizational behavior perspective [11, 12, 13, 14]. The third cluster covers project management statistics, coordination overhead, and responsibility assignment [1, 2, 15, 16].

The primary empirical material consists of two product programs executed at hh.ru between 2021 and 2026. hh.ru is Russia's leading employment platform, with more than 2,000 employees and approximately 200 product initiatives active per year. The company uses an agile delivery model in which autonomous product teams own the full stack of their product area. The author participated in both programs in the role of project manager, with inter-team dependency management formally listed as a job responsibility, a fact documented in the applicable job description.

Program A (voice calls feature) involved three development teams, one product owner, and one technology director. The scope was the integration of in-platform voice calls between employers and job seekers. Program B (contact-

center) involved five development teams, three product owners, an analytics specialist, and two leaders with non-overlapping but adjacent technical authority zones. Data collection for both programs comprised project retrospective notes, recordings of synchronization meetings and demos, Jira dependency logs. Program-level metrics (delivery timeline, feature adoption rates, award nomination outcome) were obtained from company records.

The analysis proceeds in three steps. First, a typology of inter-team interfaces is developed inductively from the case material and validated against the literature. Second, a comparative analysis applies the typology to both programs to quantify the number and type of interfaces in each. Third, the Interface Design Protocol is constructed deductively from the typology and the existing literature on coordination mechanism design [3, 5, 9], then tested for face validity against the case material.

**Results and Discussion**

A foundational assumption of agile project management is that team-internal performance, measured by metrics such as velocity and throughput, is the primary lever for program-level outcomes. This assumption is only partially supported by the literature. Berntzen et al. [3] conducted a 1.5-year ethnographic study in a large-scale agile organization and found that inter-team coordination was consistently rated as more difficult than intra-team work, irrespective of the maturity level of individual teams. Dingsøyr et al. [6] similarly report that coordination of knowledge work across teams in a 12-team program was the most time-consuming managerial activity, consuming roughly 30% of program management effort. These findings align with the theoretical prediction derived from Brooks's Law [17]: as the number of independent agents in a project grows, the number of required communication channels grows combinatorially as n(n-1)/2, not linearly.

The practical implication is that a project with three teams requires three inter-team channels, while a project with five teams requires ten. If each channel generates some fixed overhead per iteration (blocked tasks, alignment meetings, escalated decisions), the total coordination tax grows roughly 3.3 times faster than team count. This is illustrated in Figure 1.

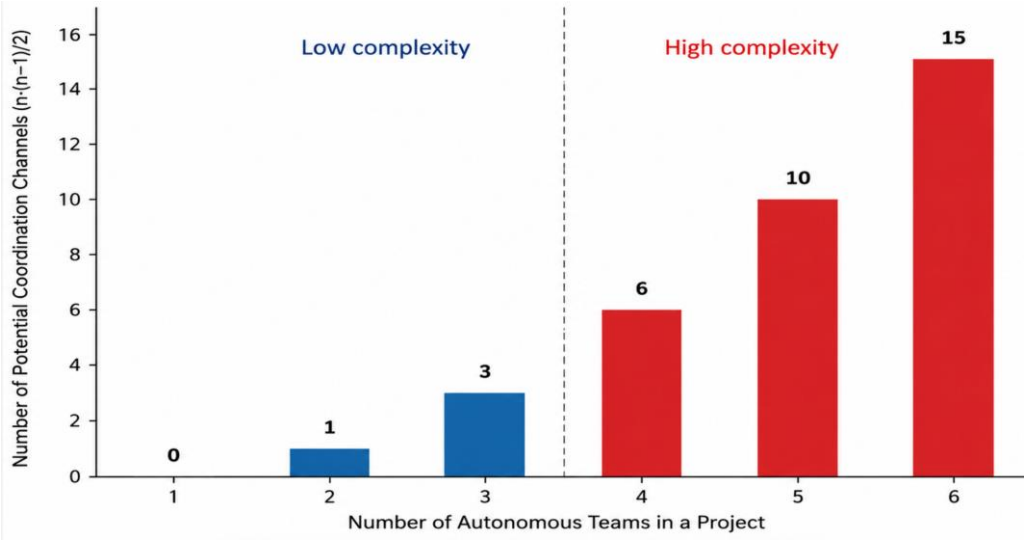



**Figure 1. Non-Linear Growth of Coordination Channels with Increasing Number of Autonomous Teams (author's calculation based on [17]; coordination channel formula n(n-1)/2).**

In Program A at hh.ru, the three-team configuration produced three horizontal coordination channels and two vertical channels, totaling five managed interfaces. In Program B, the five-team configuration produced ten horizontal channels, five vertical channels to product ownership, and two cross-perimeter channels between two leaders, totaling seventeen managed interfaces, more than three times as many. iteration retrospective data from Program B indicate that between 18% and 24% of story-point capacity per iteration was lost to blocked tasks waiting for cross-team decisions, compared with an estimated 7% to 11% in Program A, consistent with the channel-count ratio.

Table 1 summarizes the structural differences between the two programs.

**Table 1. Structural Comparison of the Two hh.ru Case Programs (compiled by the author based on program documentation and structured recall interviews (hh.ru, 2021-2024) [4, 6]).**

| Parameter | Program A: Voice Calls Feature | Program B: Contact Center |
|---|---|---|
| Development teams | 3 | 5 |
| Total coordination interfaces (calculated) | 5 | 17 |
| Delivery timeline | 6 months (on schedule) | Ongoing; first pilot at 20% employer coverage (target: 100% by 2026) |
| Estimated iteration capacity lost to coordination (retrospective) | 7-11% per iteration | 18-24% per iteration |
| Company recognition | Top 18 of ~200 annual projects | Under evaluation |
| In-platform user contact rate (feature KPI) | 56% of users making contact within 24 hours | Pilot data pending full rollout |

The 56% within-24-hour contact rate observed for voice calls is particularly striking when set against the industry-baseline statistic that only 29% of Russian residents answer calls from unknown numbers (VCIOM, 2023). This gap of 27 percentage points indicates that in-platform calls are substantially more trusted and therefore more effective than traditional phone contact, a product-level achievement that was only realizable because the three-team program maintained low enough coordination overhead to deliver on its roadmap without compressing quality.

Figure 2 contextualizes these observations against broader industry data on project outcomes by delivery model.

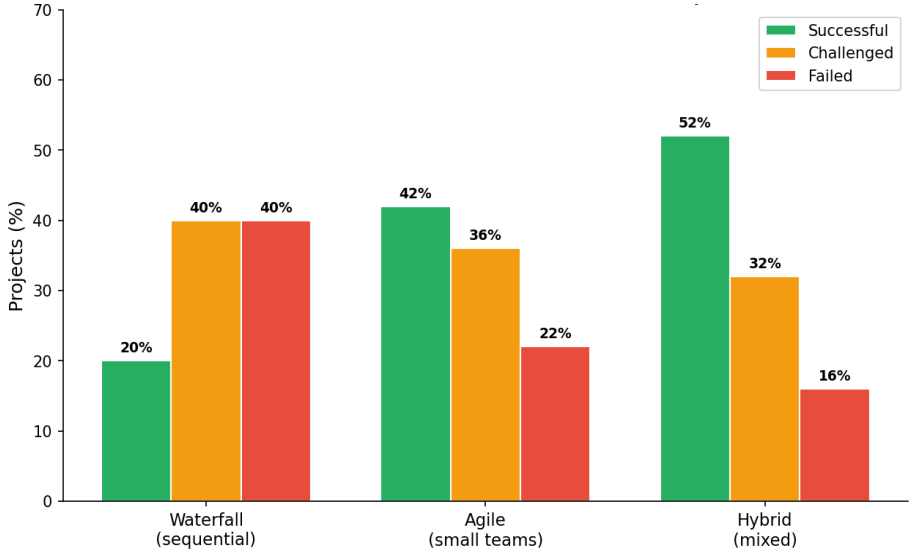



**Figure 2. Project Outcomes by Delivery Method (%). Source: PMI Pulse of the Profession 2024 [1]; Standish Group CHAOS Report 2023 [15] (compiled by the author based on [1, 15]).**

The data confirm that agile methods raise the probability of project success relative to waterfall approaches, but the absolute success rate (42% for agile) remains insufficient. The present paper argues that the residual 58% of challenged or failed agile projects are disproportionately attributable to inter-team coordination failures rather than

intra-team process deficiencies, because intra-team processes are precisely the domain where agile provides the most direct benefit.

The literature on inter-team coordination has produced several taxonomies of coordination mechanisms (meetings, roles, artifacts per [3]; standardization, supervision, mutual adjustment per [13]). However, existing taxonomies classify the mechanisms used to manage interfaces rather than the interfaces themselves. The present study proposes a typology of the interfaces as structural objects, independent of the mechanisms deployed to manage them. This distinction matters because the appropriate management mechanism depends on the type of interface, and because not all interface types are equally costly.

The typology distinguishes three interface types, illustrated in Figure 3.

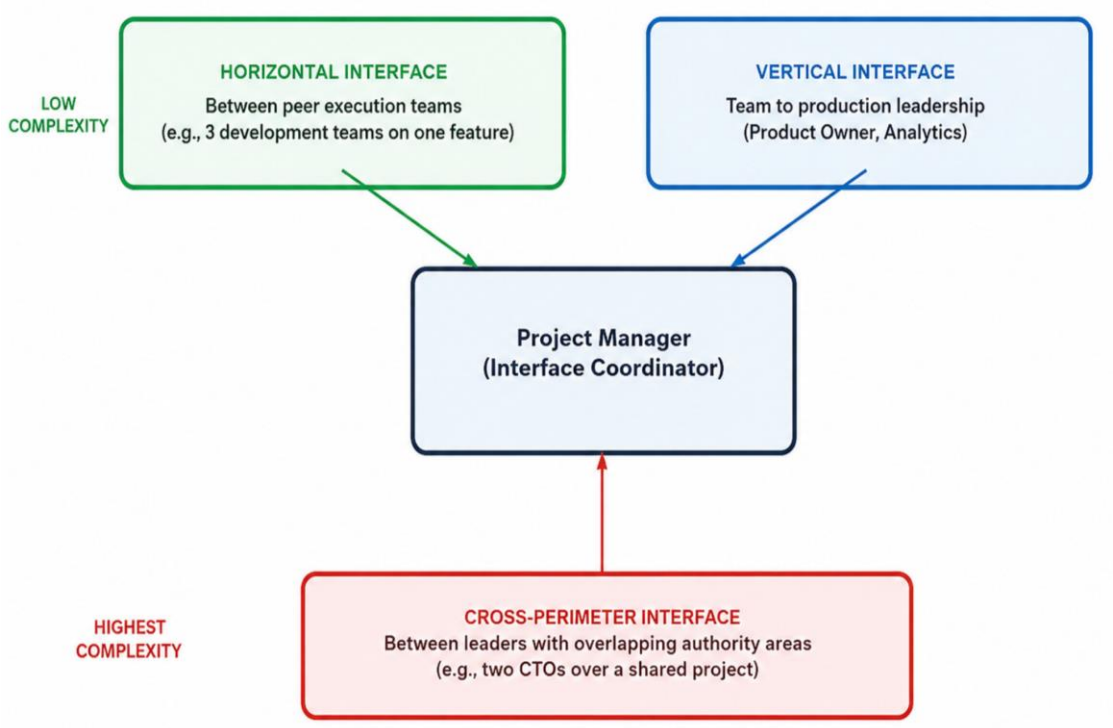



**Figure 3. Three-Level Typology of Inter-Team Interfaces (author's development based on [3, 5, 7, 18, 19]).**

Horizontal interfaces connect two teams at the same organizational level that share a task or deliver into one another's backlog. The primary coordination challenge is aligning work content: what one team produces must match what another team expects to consume. In Program A, for instance, the backend team building the call-connection protocol had to coordinate daily with the mobile team integrating the call UI, because a protocol change immediately broke the mobile build. Horizontal interfaces are frequent and high-bandwidth but comparatively tractable because both parties share similar professional language and incentives.

Vertical interfaces connect a development team to product or technical leadership (product owners, analytics, technology directors). The primary coordination challenge is translation: team-level technical decisions must be expressed in terms of business value, and business priorities must be expressed in terms of technically actionable backlog items. The asymmetry of language and incentive between the parties makes vertical interfaces more costly per unit of coordination than horizontal ones, but their number is bounded by the number of leadership roles in the program.

Cross-perimeter interfaces connect two leaders whose authority zones overlap on at least one domain without one having formal precedence over the other. This is the most costly interface type because a disagreement cannot be resolved by a working process; it requires a negotiation about authority itself. In Program B, two leaders had adjacent but distinct technical ownership scopes, and several architectural decisions fell precisely on the boundary between their domains. Each such decision required escalation to a joint session, consuming days of calendar time and blocking dependent team tasks in the interim. The cross-perimeter interface is qualitatively different from the other two types: it is not merely more expensive, it generates a category of cost (authority negotiation) that the other interface types do not.

**Table 2 provides a comparative characterization of the three interface types.**

| Attribute | Horizontal | Vertical | Cross-Perimeter |
|---|---|---|---|
| Parties | Two peer teams | Team and leadership | Two leaders, overlapping authority |
| Primary coordination challenge | Work-content alignment | Language/priority translation | Authority negotiation |
| Bandwidth (interactions/iteration) | High | Medium | Low but high-latency |
| Blockage type when unmanaged | Technical rework, integration failures | Misaligned backlog priority | Stalled decisions, escalation loops |
| Resolution mechanism | Shared definition of done, API contracts | Product Increment planning, PO sessions | Explicit joint authority agreement before program start |
| Relative unit cost | Low to medium | Medium | High |
| Scalability with team count | O(n^2) growth | O(n) growth | Fixed but disproportionately costly |

This typology allows a program to be characterized not only by the number of teams but by its interface topology: the distribution of interface types and the presence or absence of cross-perimeter interfaces. Program A had no cross-perimeter interfaces. Program B had two. Given the disproportionate cost of cross-perimeter coordination, this single structural difference accounts for a substantial fraction of the observed performance gap between the two programs.

One of the most durable insights from software engineering research is that coordination costs do not grow linearly with team size.The empirical work reviewed here provides a more granular picture: the non-linearity is not merely about team members but about autonomous decision nodes. Each team is a decision node, and each connection between nodes is a potential latency point.

Berntzen et al. [3] found in their longitudinal study that the number of coordination mechanisms required grew substantially as teams were added, consistent with a super-linear scaling pattern. The study by Berntzen, Stray, and Moe [4] confirmed that as organizations grow, they continuously introduce and adjust coordination mechanisms, often reactively rather than proactively, indicating that organizations tend to discover the complexity of their interface topology after the fact.

The Leicht-Deobald et al. meta-analysis [11], synthesizing 85 primary studies covering 10,848 teams, found that boundary spanning (the activity of managing inter-team connections) has a positive main effect on team performance, but that this effect is moderated by how boundary work is internally organized. Teams in which boundary work is distributed evenly across members perform better than teams in which it falls to a single person. This finding generalizes to the program level: programs in which interface management is distributed across formally designated interface owners outperform programs in which the project manager absorbs all interface work.

Table 3 maps the interface count implications of adding teams from one to six, applying the horizontal-channel formula n(n-1)/2 and adding estimated vertical and cross-perimeter channels based on a typical governance structure.

**Table 3. Projected Interface Count by Program Size and Topology (compiled by the author based on [3, 17]; estimates for vertical and cross-perimeter channels derived from the hh.ru case programs).**

| Teams (n) | Horizontal Channels n(n-1)/2 | Vertical Channels (est.) | Cross-Perimeter (est.) | Total Interfaces | Relative Coordination Load (index) |
|---|---|---|---|---|---|
| 2 | 1 | 2 | 0 | 3 | 1.0 |
| 3 | 3 | 3 | 0 | 6 | 2.0 (Program A) |
| 4 | 6 | 4 | 1 | 11 | 3.7 |
| 5 | 10 | 5 | 2 | 17 | 5.7 (Program B) |
| 6 | 15 | 6 | 3 | 24 | 8.0 |

The index in the final column is normalized to the two-team baseline. By this estimate, moving from three teams (Program A) to five teams (Program B) does not double coordination load but increases it by a factor of 2.85. The presence of cross-perimeter interfaces accelerates this growth because cross-perimeter decisions function as blockers rather than delays: they cannot proceed at all until authority is negotiated, whereas horizontal and vertical decisions can often proceed with partial information.

For US-based technology organizations operating at scale, this analysis has direct budgetary significance. If each coordination channel generates an average of one blocked story per iteration, and if the average story takes one engineering day to complete once unblocked, a five-team program with 17 interfaces loses 17 engineering days per iteration to coordination alone, roughly 4.25 engineering weeks per iteration or the equivalent of one full-time engineer continuously engaged in rework caused by interface misalignment.

The core recommendation of this paper is that inter-team interfaces should be explicitly designed before a program begins, in the same way that software architecture is designed before coding starts. The proposed framework is the Interface Design Protocol (IDP). It consists of five sequenced steps applied during program scoping, before the first iteration. The sequence is depicted in Figure 4.

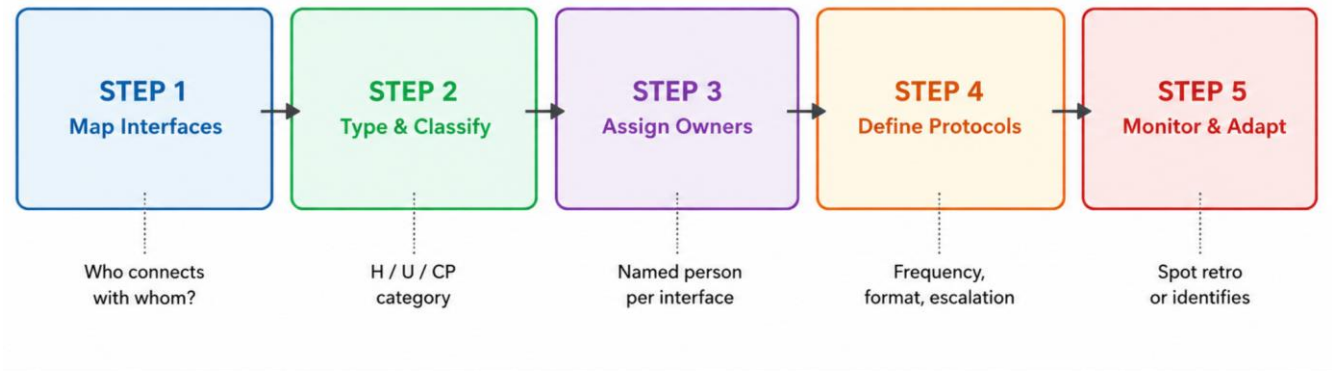



**Figure 4. Interface Design Protocol (IDP): Sequence of Pre-Launch Actions (author's development).**

Step 1 (Map Interfaces) requires the program manager to enumerate every pair of teams or leaders that will need to exchange decisions or deliverables. This mapping should be done as a structured workshop with all team leads and is complete when every dependency in the program backlog can be traced to a named interface.

Step 2 (Type and Classify) applies the typology to each identified interface. The classification determines the appropriate management mechanism and the priority level for pre-launch investment. Cross-perimeter interfaces

require authority resolution before the program starts; horizontal interfaces require shared technical contracts; vertical interfaces require agreed translation rituals (planning sessions, demo cadence).

Step 3 (Assign Owners) names a specific person as the owner of each interface. The owner's responsibility is not to perform all coordination work personally but to ensure that the interface does not become a bottleneck. This is consistent with the RACI-matrix principle [16] applied at the interface level rather than the task level. In programs with many horizontal interfaces, ownership should be distributed across team leads rather than concentrated in the project manager, consistent with the finding of Leicht-Deobald et al. [11] that distributed boundary work outperforms centralized boundary work.

Step 4 (Define Protocols) specifies for each interface the synchronization frequency (daily, per iteration, per program increment), the communication format (shared artifact, standing meeting, ad-hoc escalation), and the escalation path when the interface becomes blocked. For cross-perimeter interfaces, the escalation path must be defined before the program starts, because ad-hoc escalation is precisely the pattern that generates the longest delays.

Step 5 (Monitor and Adapt) embeds interface health into the program's regular retrospectives. Each iteration retrospective should include a standing agenda item reviewing which interfaces generated blockages, how quickly they were resolved, and whether the type classification or ownership assignment needs revision. The Berntzen et al. [4] longitudinal study demonstrates that organizations that monitor and adapt coordination mechanisms over time achieve better outcomes than those that treat coordination design as a one-time activity.

The IDP is not a new meeting cadence or a communication tool. It is a structural intervention that precedes communication: it determines who needs to communicate with whom, about what, and by what authority, before any communication happens. This distinction is operationally significant. Many programs experience coordination failures not because team members fail to communicate but because they communicate about the wrong things or wait for the wrong person's decision. The IDP eliminates that structural ambiguity by making it explicit.

In the hh.ru voice-call program (Program A), the absence of cross-perimeter interfaces and the early establishment of shared technical contracts between teams approximated several IDP elements informally. In the contact-center program (Program B), the cross-perimeter interfaces between the two leaders were not formally mapped until the program was already underway, and the first authority negotiation between them took approximately three weeks, blocking two of the five development teams during that period. A pre-launch IDP session would have surfaced this interface type, prompted an authority agreement before the teams started work, and eliminated the blockage [20, 21].

This analysis supports the paper's central hypothesis: coordination losses in large-scale projects are not primarily caused by poor communication but by absent structural design of the channels through which communication would need to flow. The IDP converts an informal, reactive activity into a formal, proactive discipline, which is the appropriate response to a structural problem.

## Conclusion

This paper has argued, supported, and illustrated three related claims. First, in multi-team projects, the dominant source of schedule and resource losses is not within-team inefficiency but the coordination overhead generated at inter-team interfaces, a finding consistent with the empirical literature on large-scale agile development and substantiated by comparing two hh.ru product programs with different interface topologies. Second, inter-team interfaces are not homogeneous: the proposed three-level typology distinguishes horizontal interfaces (between peer teams), vertical interfaces (between teams and leadership), and cross-perimeter interfaces (between leaders with overlapping authority), each with a distinct cost profile and management requirement. Cross-perimeter interfaces are qualitatively different because they generate authority-negotiation costs rather than mere communication costs, and their presence in Program B explains a substantial portion of the performance gap relative to Program A. Third, the Interface Design Protocol (IDP) is proposed as a practical pre-launch discipline that makes interface mapping, classification, ownership assignment, protocol definition, and retrospective monitoring explicit managerial activities rather than emergent, informal ones.

The study achieves the goal stated in the abstract: it identifies the mechanism by which inter-team interfaces generate losses, proposes a typology that allows their classification, and offers a practitioner-ready framework for managing them proactively. The empirical contrast between two programs at the same company, with comparable technical

scope but starkly different interface topologies, provides an illustrative case for the argument and suggests that the structural variable of interface count and type predicts coordination overhead more reliably than team-internal maturity alone.

The practical significance of these findings is highest for organizations operating at the three-to-six-team program scale, where the non-linearity of coordination channel growth has the sharpest inflection (Table 3) and where cross-perimeter interfaces are most likely to appear without being explicitly designed. For project managers and program directors, the IDP offers a concrete, session-based tool that fits within existing pre-launch planning activities. For HR practitioners involved in team structure decisions, the typology provides a lens for evaluating whether a proposed team topology will generate manageable or unmanageable coordination demands before the program begins.

Future research should validate the IDP in a prospective controlled setting, formally measuring coordination overhead before and after its application. Quantitative studies correlating interface type distribution with iteration velocity data across a larger sample of programs would test the non-linear growth hypothesis with greater statistical power. The cross-perimeter interface type, identified here as the most costly, deserves particular attention: its prevalence in matrix organizations and dual-leadership structures makes it a high-priority topic for both research and practitioner guidance.

## References


1. Project Management Institute. (2024, February). Pulse of the Profession® 2024: The future of project work. Retrieved January 19, 2026, from https://www.pmi.org/learning/thought-leadership/future-of-project-work

2. McKinsey & Company. (2023, August 1). The state of AI in 2023: Generative AI's breakout year. Retrieved February 16, 2026, from https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year

3. Berntzen, M., Hoda, R., Moe, N. B., & Stray, V. (2023). A taxonomy of inter-team coordination mechanisms in large-scale agile. IEEE Transactions on Software Engineering, 49(2), 699–718. https://doi.org/10.1109/TSE.2022.3160873

4. Berntzen, M., Stray, V., Moe, N. B., & Hoda, R. (2023). Responding to change over time: A longitudinal case study on changes in coordination mechanisms in large-scale agile. Empirical Software Engineering, 28(5), Article 114. https://doi.org/10.1007/s10664-023-10349-0

5. Berntzen, M., Stray, V., & Moe, N. B. (2021). Coordination strategies: Managing inter-team coordination challenges in large-scale agile. In P. Gregory, C. Lassenius, X. Wang, & P. Kruchten (Eds.), Agile processes in software engineering and extreme programming (Lecture Notes in Business Information Processing, Vol. 419, pp. 140–156). Springer. https://doi.org/10.1007/978-3-030-78098-2_9

6. Dingsøyr, T., Bjørnson, F. O., Schrof, J., & Sporsem, T. (2023). A longitudinal explanatory case study of coordination in a very large development programme: The impact of transitioning from a first- to a second-generation large-scale agile development method. Empirical Software Engineering, 28(1), Article 1. https://doi.org/10.1007/s10664-022-10230-6

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, Article 111649. https://doi.org/10.1016/j.jss.2023.111649

8. Yin, R. K. (2018). Case study research and applications: Design and methods (6th ed.). SAGE Publications.

9. Vedal, H., Stray, V., Berntzen, M., & Moe, N. B. (2021). Managing dependencies in large-scale agile. In P. Gregory & P. Kruchten (Eds.), Agile processes in software engineering and extreme programming – Workshops (Lecture Notes in Business Information Processing, Vol. 426, pp. 52–61). Springer. https://doi.org/10.1007/978-3-030-88583-0_6

10. Dingsøyr, T., Moe, N. B., & Seim, E. A. (2018). Coordinating knowledge work in multiteam programs: Findings from

a large-scale agile development program. Project Management Journal, 49(6), 64–77. https://doi.org/10.1177/8756972818798980

**11.** Leicht-Deobald, U., Backmann, J., de Vries, T. A., Weiss, M., Hohmann, S., Walter, F., van der Vegt, G. S., & Hoegl, M. (2025). A contingency framework for the performance consequences of team boundary management: A meta-analysis of 30 years of research. Journal of Management, 51(2), 704–747. https://doi.org/10.1177/01492063231206107

**12.** Rodriguez-Escudero, A. I., Carbonell, P., Perez-Fernandez, H., & Temprano-Garcia, V. (2025). Examining the role of team boundary buffering as a proactive and reactive stress intervention for boundary spanning new product teams. Journal of Engineering and Technology Management, 75, Article 101861. https://doi.org/10.1016/j.jengtecman.2025.101861

**13.** Wagner, J. A., III. (2024). Inter-team coordination in multiteam systems: Mechanisms, transitions, and precipitants. Organizational Psychology Review, 14(3), 425–448. https://doi.org/10.1177/20413866231153537

**14.** Leicht-Deobald, U., Lam, C. F., Bruch, H., Kunze, F., & Wu, W. (2022). Team boundary work and team workload demands: Their interactive effect on team vigor and team effectiveness. Human Resource Management, 61(4), 465–488. https://doi.org/10.1002/hrm.22104

**15.** Standish Group International. (2020). CHAOS report: Beyond infinity. Retrieved May 27, 2026, from https://www.standishgroup.com/products/copy-of-chaos-report-beyond-infinity-digital-version

**16.** Project Management Institute. (2021). The standard for project management and a guide to the project management body of knowledge (PMBOK® guide) (7th ed.).

**17.** Gote, C., Mavrodiev, P., Schweitzer, F., & Scholtes, I. (2022). Big data = big insights? Operationalising Brooks' law in a massive GitHub data set. In Proceedings of the 44th International Conference on Software Engineering (pp. 262–273). Association for Computing Machinery. https://doi.org/10.1145/3510003.3510619

**18.** Biesialska, K., Franch, X., & Muntés-Mulero, V. (2021). Mining dependencies in large-scale agile software development projects: A quantitative industry study. In Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE 2021) (pp. 20–29). Association for Computing Machinery. https://doi.org/10.1145/3463274.3463323

**19.** Strode, D., Dingsøyr, T., & Lindsjørn, Y. (2022). A teamwork effectiveness model for agile software development. Empirical Software Engineering, 27, Article 56. https://doi.org/10.1007/s10664-021-10115-0

**20.** Chang, J. Y. T., Wu, X., Discenza, R., & Klein, G. (2022). Creative software development with team boundary management. Journal of Computer Information Systems, 62(4), 694–705. https://doi.org/10.1080/08874417.2021.1903366

**21.** Wu, X., Tsai, J. C.-A., & Lei, Y. (2023). Information technology project management research: A review of works by influential pioneers. Project Management Journal, 54(4), 366–391. https://doi.org/10.1177/87569728231171056