World Journal of Advanced Research and Reviews

eISSN: 2581-9615 CODEN (USA): WJARAI

Cross Ref DOI: 10.30574/wjarr

Journal homepage: https://wjarr.com/

(Review Article)

A process model for distributed engineering development in international teams

Ilia Titovskii *

An expert in software engineering and digital product development. Russia.

World Journal of Advanced Research and Reviews, 2025, 25(03), 2534-2537

Publication history: Received on 02 February 2025; revised on 21 March 2025; accepted on 29 March 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.25.3.0741

Abstract

Distributed development in international teams increases systemic risks: goal desynchronization, context loss, longer delivery cycles, increased defect rates at team interfaces, and decreased release predictability. However, a distributed format can scale product development if engineering processes are restructured for asynchrony, decision transparency, and measurable quality. This article proposes a process model for distributed engineering teams: (i) a specification and decision loop (RFC/ADR), (ii) a delivery loop (CI/CD and quality gates), (iii) a communication loop (synchronization rituals), (iv) a responsibility loop (ownership and SLAs), (v) a feedback loop (observability, incidents, learning). It also demonstrates how to balance synchronous and asynchronous practices based on overlapping time zones and how to translate "teamwork" into a reproducible operating system.

Keywords: Distributed Teams; Engineering Processes; Asynchronous Communication; RFC; ADR; CI/CD; Quality; Ownership; SLO/SLA; Observability; Knowledge Management.

1. Introduction

International distributed development has become a stable norm for digital products: teams are distributed across countries, legal zones, and time zones, creating a broader talent pool and increasing business resilience. However, the benefits of distribution only emerge when the organizational fabric supports contextual communication and controlled decision-making. Otherwise, transaction costs increase: communication becomes more expensive, decisions slower, and errors become more systemic.

The effects of distribution are especially noticeable in teams with diverse cultural and operational contexts: differences in working hours, cultural communication habits, expectations of documentation, and release discipline. The engineering challenge is to transform these differences from a source of chaos into a manageable model: to establish a minimal set of rules where processes become "system memory" and reduce dependence on the presence of specific individuals at a specific time and place.

1.1. The Problem of Distributed Design: What Exactly Breaks

In distributed teams, five characteristics most often degrade:

- Decision-making speed (waiting for responses, approval chains).
- Contextual integrity (why a decision was made, which alternatives were rejected).
- Integration quality (defects at the interfaces of services and components).
- Release predictability (coordination complexity, release windows).
- Reliability and incident response (the question "Who is the owner?" and "When is they available?").

* Corresponding author: Ilia Titovskii

This leads to the basic principle: processes should compensate not for "remoteness," but for loss of context and delays.

1.2. Balancing synchronous and asynchronous practices

Synchronous meetings (stand-ups, planning sessions, retrospectives) are important for aligning priorities, but their effectiveness decreases with low time overlap. Asynchronous practices (RFCs, written decisions, reviews, comments), on the contrary, benefit as overlap decreases, but require documentation discipline and a uniform format.

Conceptually, dependency can be represented as follows:

1.2.1. The problem of distribution: what exactly breaks

- In distributed teams, five characteristics most often degrade:
- Decision-making speed (waiting for responses, approval chains).
- Contextual integrity (why a decision was made, which alternatives were rejected).
- Integration quality (defects at the interfaces of services and components).
- Release predictability (coordination complexity, release windows).
- Reliability and incident response (the question "who is the owner?" and "when are they available?").

This leads to the basic principle: processes should compensate not for "remoteness," but for loss of context and delays.

1.2.2. Balancing Synchronous and Asynchronous Practices

Synchronous meetings (stand-ups, planning, retros) are important for aligning priorities, but their effectiveness declines with low time overlap. Asynchronous practices (RFCs, written solutions, reviews, comments) benefit as overlap decreases, but require documentation discipline and a consistent format.

Conceptually, this relationship can be represented as follows:

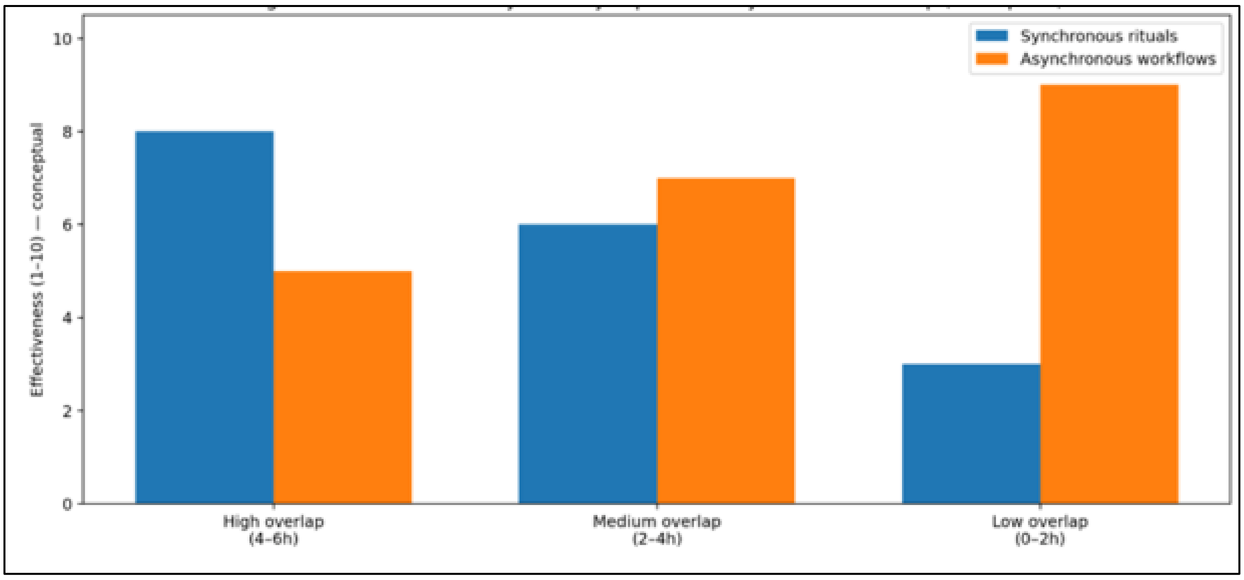

Figure 1 Effectiveness of sync vs async processes by time zone overlap

Practical conclusion: the less overlap in working hours, the more the process should rely on "written context transfer" and standardized artifacts (specs, ADRs, quality checklists).

1.2.3. Documentation as infrastructure: RFCs and ADRs

Two forms of artifacts are most useful in distributed development:

RFCs (Request for Comments): a structured change specification (problem, requirements, options, risks, implementation plan, success metrics).

ADRs (Architecture Decision Records): a short recording of the architectural decision (context → decision → consequences).

They solve a fundamental problem: the team stops "remembering" decisions in their heads and transfers them to the system. This reduces dependence on synchronous meetings and simplifies onboarding. Importantly, RFCs/ADRs should be lightweight and regular, otherwise documentation becomes a barrier.

1.2.4. Ownership and Responsibility Frameworks

With distributed systems, the "nobody's right" error becomes systemic: if it's unclear who owns a component, incidents drag on, and changes are made through informal channels. Therefore, the following are introduced:

- Ownership by service/module (explicit owners),
- Responsibility matrix (who makes decisions, who reviews, who is on duty),
- Responsibility slas (internal expectations: response/escalation time),
- Escalation rules (what to do if a block occurs during the other party's "non-working" hours).

In an international team, this is especially critical: part of the team may be unavailable when the issue occurs, so the process must determine in advance who assumes responsibility.

1.2.5. Delivery as "quality control by default": CI/CD and quality gates

In a distributed model, quality can't be "verbally negotiated": it must be enshrined in the delivery pipeline. Therefore, automated "quality gates" are being strengthened:

- Static analysis and linting,
- Unit/integration tests as a minimum standard,
- Api contract verification (contract testing),
- Review policies (minimum 1-2 reviewers, test requirements),
- Safe release strategies (feature flags, gradual rollout, rapid rollback).

The idea is simple: when communication is more expensive, defects must be caught early and automatically, otherwise the cost of fixes increases disproportionately.

1.2.6. Observability and Incidents: A Common Feedback Language

Distributed teams benefit when they rely on measurable signals rather than just "feelings":

- Availability and latency metrics (SLO),
- Tracing for inter-service chains,
- Structured logs,
- Client metrics (for mobile/frontend).

Incident management should be standardized: who is on duty, how the timeline is recorded, how the postmortem is conducted. A postmortem in a distributed environment is not a formality, but a learning mechanism that replaces the "random transfer of knowledge."

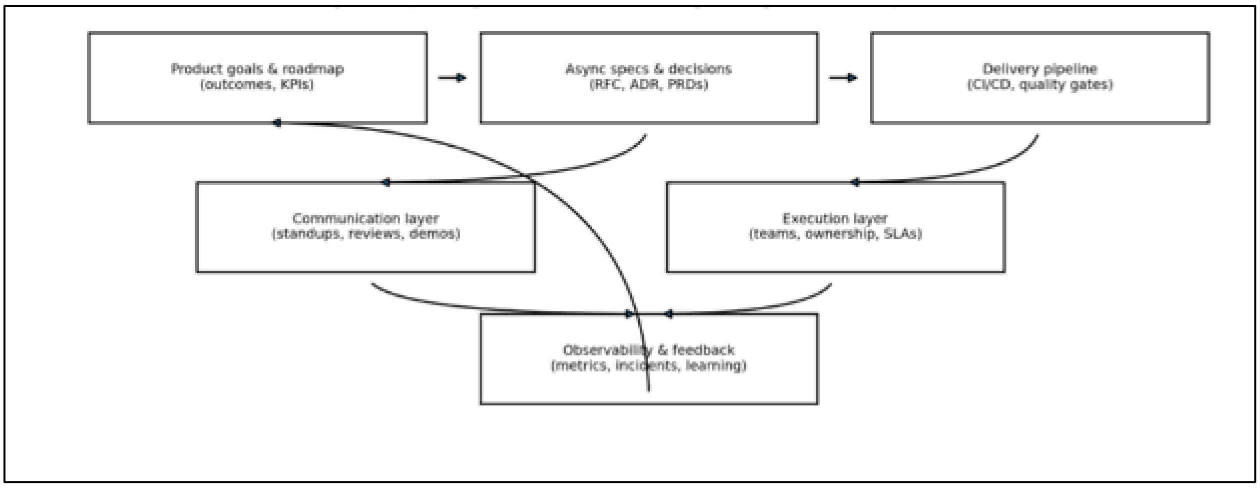

Figure 2 Operating model for distributed engineering teams

1.2.7. Distributed Team Operating Model

Processes are effective when they form a closed loop: goals → decisions → delivery → execution → feedback → goal adjustment.

This allows the team to be resilient to personnel changes and geographical distribution: the system operates even when individual members are temporarily unavailable.

2. Conclusion

Distributed development in international teams requires a shift from "communication by default" to "processes by default." Key elements include asynchronous artifacts (RFC/ADR), transparent accountability (ownership and SLAs), automated quality gates (CI/CD), and observability as a common language of facts. The balance of synchronous and asynchronous practices should be determined by time zone overlap: the smaller the overlap, the greater the value of written communication of context and standardization of solutions. As a result, distribution ceases to be a source of waste and becomes a scalable operating model that supports the speed and quality of product development.

References

[1] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. Portland, OR, USA: IT Revolution Press, 2018.

[2] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, eds., Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA, USA: O'Reilly Media, 2016.

[3] M. T. Nygard, Release It!: Design and Deploy Production-Ready Software, 2nd ed. Dallas, TX, USA: Pragmatic Bookshelf, 2018.

[4] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol, CA, USA: O'Reilly Media, 2017.

[5] M. Fowler, "Architecture Decision Records," martinfowler.com, 2017. [Online]. Available: https://martinfowler.com/articles/architectural-records.html. Accessed: Feb. 16, 2026.