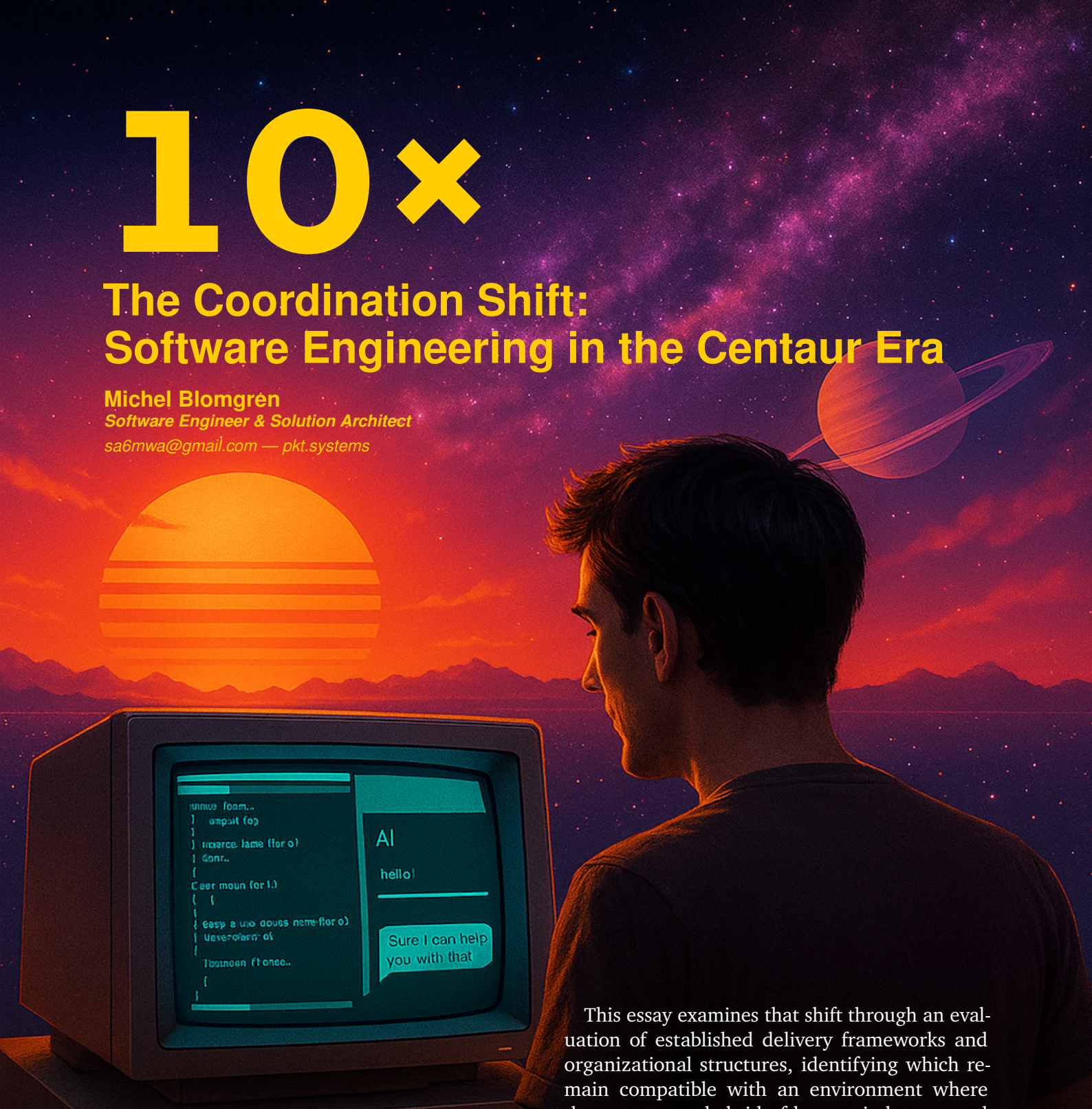
10×

The Coordination Shift: Software Engineering in the Centaur Era

Michel Blomgren
Software Engineer & Solution Architect
sa6mwa@gmail.com — pkt.systems



November 26, 2025

The emergence of capable agentic AI systems has begun to shift the practical boundaries of software engineering. Senior engineers, working in partnership with AI coding agents, can now deliver systems whose scope and coherence previously demanded coordinated human teams. This transformation alters the economics of software work: execution accelerates, iteration becomes inexpensive, and coordination within the human–AI unit largely disappears. The central constraints shift outward, toward governance, architectural boundaries, and inter-unit alignment.

This essay examines that shift through an evaluation of established delivery frameworks and organizational structures, identifying which remain compatible with an environment where the centaur—a hybrid of human judgment and machine-driven execution—emerges as a new unit of production. The analysis is illustrated by `lockd`, a non-trivial coordination service created by a senior engineer in tandem with an AI agent in roughly five weeks of spare time, demonstrating the practical viability of this mode of work.

The resulting picture suggests a reorientation of software engineering around autonomy, modularity, and automated governance, with coordination—not implementation—becoming the primary bottleneck in increasingly centaur-shaped organizations.

INTRODUCTION

For most of the history of software engineering, the fundamental constraint has been human cognitive capacity. Classic accounts of software projects describe coordination overhead, limited communication bandwidth, and the difficulty of maintaining shared understanding as primary bottlenecks (Brooks, 1995; Conway, 1968). Delivery frameworks and organizational models—from plan-driven methods to agile and DevOps practices—can be understood as structured responses to those constraints, seeking to economize on scarce human attention and to manage the flow of information through teams and hierarchies (Cockburn, 2001; Highsmith, 2001; Humble, Molesky, and O'Reilly, 2015; Galbraith, 1974; Burns and Stalker, 1961).

Around 2025, a qualitatively different situation emerged. A new class of large-language-model-driven development environments and agentic workflows made it practically feasible for experienced engineers to offload substantial portions of design, coding, testing, and operational tasks to AI systems, without spending more time correcting errors than the tools saved. Empirical studies of AI-assisted programming already indicate significant productivity gains for certain task types, alongside mixed effects on code quality and security (Peng, 2023; Hamza et al., 2023; Ercin, 2025; Oladele and Lawal, 2025; Schreiber, 2025). At the same time, research on human–AI collaboration and AI "teammates" highlights novel coordination challenges, including trust calibration, responsibility attribution, and the management of opaque model behaviour (Seeber, 2020; Shneiderman, 2020; Schmutz, 2024).

This paper takes that post-2025 reality as a starting point. We[1] assume a *centaur* model of software development in the spirit of Kasparov's advanced chess: a human expert working in tight cognitive partnership with machine agents, where the human provides framing, judgment, and accountability, and the AI systems provide search, execution, and rapid exploration (Kasparov, 2017; Alves and Cipriano, 2023). In this setting, a single senior engineer, augmented by such tools, can frequently execute end-to-end work—architecture, implementation, testing, and operations—that previously required a coordinated team. Crucially, this is not a claim about the disappearance of human labor or the obsolescence of teams, but about a changed *unit of effective action*: a centaur-like human–AI unit can now perform work packages that most existing methods were designed to allocate to multi-person groups.

Most established delivery frameworks and organizational structures were not designed with this unit in mind. Scrum, Extreme Programming, Kanban, scaled frameworks such as SAFe and LeSS, and outcome-oriented approaches such as Lean Startup all embody particular assumptions about how work is discovered, sliced, assigned, and synchronized in human-only teams (Beck, 1999; Anderson, 2010; Scaled Agile Inc., 2021; Larman and Vodde, 2017; Ries, 2011). Organizational designs such as functional hierarchies, product divisions, matrix structures, and modern team-based configurations similarly encode trade-offs between centralization, local autonomy, and information-processing capacity (Mintzberg, 1979; Galbraith, 1974; Senge, 1990; Skelton and Pais, 2019). None of these bodies of work explicitly anticipates a world in which an individual senior engineer plus AI agents can act, for many purposes, as a "micro-team" with its own exploratory and delivery capability.

THE AIM of this exploratory essay is to examine how frameworks and organizational models align with a centaur development reality. We do not attempt a comprehensive empirical evaluation; quantitative evidence on AI-augmented development remains sparse and rapidly evolving. Instead, we combine (i) established research on coordination, organizational design, and software delivery performance; (ii) emerging findings on AI-assisted programming, security, and human–AI collaboration; and (iii) practitioner observations from projects by senior engineers working with agentic tools. On this basis, we propose evaluation dimensions for delivery frameworks and organizational models, and assess their compatibility with individual-plus-AI development. The analysis brackets "replacement" narratives: the question is not whether AI will substitute for developers, but how human work, coordination structures, and governance must evolve when a single senior centaur unit can do what once required an entire team.

[1] *"We"* is intentional as the essay was written by a centaur unit.

BACKGROUND

COORDINATION AND THE EVOLUTION OF SOFTWARE METHODS

Software engineering methods have historically emerged as responses to coordination constraints. Early observations by Brooks (Brooks, 1995) and Conway (Conway, 1968) highlighted the communication overhead inherent in multi-person development and the structural coupling between organizational boundaries and software architectures. Subsequent research across organizational theory reinforced these findings: Galbraith's information-processing view of organizations (Galbraith, 1974), Mintzberg's structural configurations (Mintzberg, 1979), and Burns and Stalker's analysis of mechanistic versus organic forms (Burns and Stalker, 1961) all frame coordination capacity, not individual productivity, as the limiting factor in complex environments.

Process models and delivery frameworks can thus be understood as mechanisms to manage these structural constraints. Plan-driven approaches formalized control through sequential stages and documentation; agile methods shifted the emphasis toward short feedback cycles, shared understanding, and direct collaboration (Cockburn, 2001; Highsmith, 2001). Practices such as Scrum (Schwaber and Sutherland, 2017) and Extreme Programming (Beck, 1999) sought to minimize misalignment through stable teams, bounded work-in-progress, and iterative planning. DevOps and Lean Enterprise variants (Humble, Molesky, and O'Reilly, 2015; Forsgren, Humble, and Kim, 2018) extended these ideas to bridge development and operations, emphasizing continuous delivery, telemetry, and reduction of batch sizes. Across these traditions, the core assumption is constant: software delivery is a socio-technical process in which human coordination effort is the principal bottleneck.

HUMAN–MACHINE COLLABORATION AND THE CENTAUR MODEL

The rise of machine assistance in knowledge work has long been framed through the lens of *human–machine symbiosis*. Hutchins' theory of distributed cognition (Hutchins, 1995) conceptualizes cognitive work as spanning humans, artifacts, and representational structures. Naturalistic decision-making research (e.g., Klein (Klein, 2008)) highlights how experts integrate environmental cues, pattern recognition, and iterative reframing in high-uncertainty domains—a pattern mirrored in modern software design and troubleshooting.

Kasparov's notion of *advanced chess* introduced the "centaur" model, in which human strategic judgment combines with machine computational capacity (Kasparov, 2017). Empirical outcomes in chess demonstrated that human–machine teams could outperform both humans and machines operating alone, not due to raw computation but due to the complementary interplay between human framing and machine search. Recent accounts extend the centaur metaphor to professional work more broadly (Alves and Cipriano, 2023), arguing that the locus of expertise shifts from handcrafted output to directing, interpreting, and governing machine activity.

Parallel streams in human–AI teaming research emphasize similar dynamics. Seeber et al. (Seeber, 2020) outline coordination, trust, and responsibility as persistent challenges when AI systems act as teammates rather than tools. Work on human-centered AI (Shneiderman, 2020) stresses the importance of oversight structures and predictable interaction patterns. Emerging studies of AI-assisted programming identify both productivity benefits (Peng, 2023; Hamza et al., 2023) and new risks associated with opaque model behavior and security vulnerabilities in AI-generated code (Oladele and Lawal, 2025; Schreiber, 2025). Together, these lines of research suggest that machine assistance alters the distribution of cognitive labor but does not eliminate the need for human framing, verification, and governance.

DELIVERY FRAMEWORKS AND ORGANIZATIONAL MODELS AS COORDINATION REGIMES

Delivery frameworks and organizational structures encode assumptions about how coordination should occur. Scrum and related agile methods assume stable teams, frequent synchronous communication, and shared responsibility for product increments (Schwaber and Sutherland, 2017). Kanban emphasizes flow efficiency and adaptive pull-based coordination (Anderson, 2010). Scaled frameworks such as SAFe (Scaled Agile Inc., 2021) and LeSS (Larman and Vodde, 2017) introduce multi-team synchronization mechanisms intended to preserve alignment at higher organizational layers. Lean Startup (Ries, 2011) reorients delivery around hypothesis testing and rapid experimentation, while DevOps and SRE embed delivery within socio-technical systems of automation, feedback, and monitoring (Humble, Molesky, and O'Reilly, 2015; Forsgren, Humble, and Kim, 2018).

Organizational models similarly represent coordination regimes. Functional hierarchies optimize for specialization but introduce cross-boundary dependencies; divisional structures emphasize local autonomy; matrix configurations attempt to balance expertise and product alignment (Mintzberg, 1979). Contemporary approaches such as team-based topologies (Skelton and Pais, 2019) and modular organizational forms (Sanchez and Mahoney, 1996) seek to reduce interdependence by structuring the organization into semi-autonomous units with well-defined interaction surfaces.

Across all these methods and structures, the central assumption is that coordination must be managed among groups of humans. None of them explicitly anticipate a setting in which an individual senior engineer, augmented by agentic AI systems, can perform the exploration, implementation, and operational work previously requiring a team. The centaur development reality introduced in Introduction (p. 2) therefore raises foundational questions about the continued suitability of these coordination regimes.

THE POST-2025 CENTAUR DEVELOPMENT REALITY

THE CAPABILITY INFLECTION POINT

By 2025, agentic large-language-model (LLM) systems reached a practical threshold at which experienced engineers could delegate substantial portions of software work—including design exploration, implementation, refactoring, test generation, and operational tasks—without incurring prohibitive verification or correction overhead. This shift did not arise from a single product or vendor, but from the convergence of several developments: improved context handling, more reliable tool use, iterative code-editing capabilities, and tighter integration between natural-language specifications and executable artifacts. Empirical studies of AI-assisted programming already show meaningful productivity improvements on well-scoped tasks (Peng, 2023), while also identifying limitations and risks in areas such as security, correctness, and maintainability (Ercin, 2025; Oladele and Lawal, 2025; Schreiber, 2025). These findings align with broader observations in human–AI teaming research, which emphasize that AI systems can extend human capability but also introduce new coordination demands (Seeber, 2020; Schmutz, 2024).

The significance of this inflection point lies not merely in faster code production but in the reconfiguration of the *unit of effective work*. Tasks that previously required a cross-functional team—encompassing design, coding, testing, and operational knowledge—can now be executed by a single senior engineer working in close cognitive partnership with AI agents. This does not eliminate the role of teams; rather, it changes the granularity at which teams add value. Many work items that required intra-team coordination can now be handled within a single human–AI unit, while coordination mechanisms between units, systems, and organizational boundaries remain critical.

HUMAN ROLES

Despite rapid advances in agentic tooling, human expertise remains essential for the tasks that require contextual reasoning, uncertainty management, and domain-specific judgment. Research on naturalistic decision making (Klein, 2008) and distributed cognition (Hutchins, 1995) suggests that experts excel at recognizing patterns, identifying anomalies, reframing problems, and integrating diverse cues—capabilities that AI systems do not yet replicate reliably. In the centaur development setting, senior engineers provide:

- **Problem framing and abstraction**: determining the boundaries, constraints, and intended behavior of the system.
- **Architectural and design judgment**: evaluating trade-offs, selecting structural patterns, and managing long-term consequences.
- **Verification and interpretation**: validating AI-generated artifacts, interpreting failures, and ensuring alignment with requirements.
- **Governance and ethical oversight**: maintaining security, safety, compliance, and responsible use of automation.

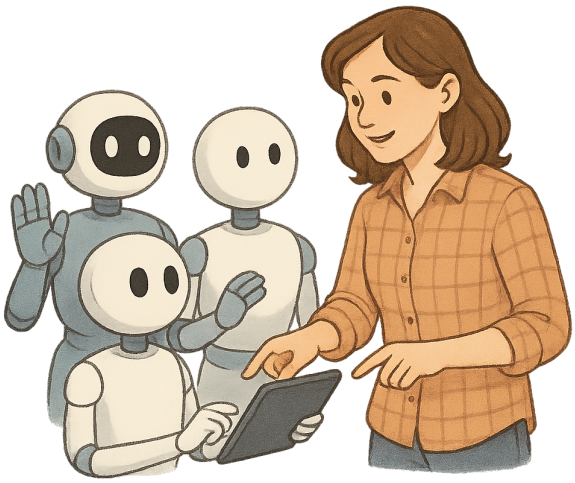
These responsibilities are amplified, not diminished, by the presence of AI agents. While AI can accelerate hypothesis generation and code synthesis, it also increases the need for systematic verification and oversight, reflecting well-known concerns in human-centered AI (Shneiderman, 2020). The centaur pattern thus resembles advanced chess (Kasparov, 2017): the human provides strategic framing and quality control, while the AI executes rapid tactical exploration.

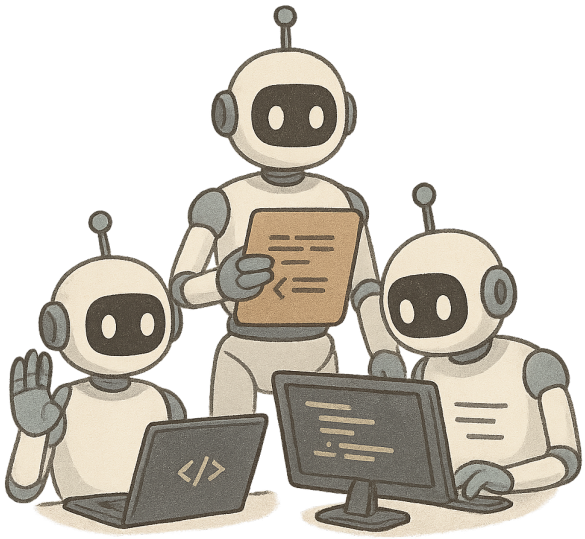
MACHINE ROLES

AI agents contribute fundamentally different capabilities from human engineers. Whereas human expertise is optimized for contextual reasoning, AI systems excel at high-throughput search, rapid transformation of code, and generating multiple candidate solutions. In software development, this includes:

- **Rapid code synthesis and refactoring**: producing initial implementations or transforming existing ones.
- **Exploratory prototyping**: generating alternative design or implementation pathways for comparison.
- **Automated test generation**: constructing test harnesses and examples that support verification.
- **Operational interactions**: issuing commands, inspecting system state, or proposing remediation steps when integrated with tools.

These capabilities reduce execution cost and increase the feasibility of high-frequency iteration—both core determinants of performance in modern delivery settings (Forsgren, Humble, and Kim, 2018). However, they do not eliminate coordination demands. AI-generated artifacts must be integrated, validated, and aligned with architectural intent, and AI-driven changes can introduce new security or reliability concerns, as recent studies highlight (Oladele and Lawal, 2025; Schreiber, 2025). Thus, machine capability shifts the distribution of labor but not the necessity of human governance.





NEW COORDINATION AND GOVERNANCE CHALLENGES

The centaur development reality introduces coordination challenges that are distinct from those addressed by existing delivery frameworks. Traditional methods assume that coordination overhead arises from communication between humans—synchronizing requirements, designs, and work across a team. In a human–AI unit, intra-unit coordination becomes comparatively cheap: the engineer and AI agents can iterate rapidly, explore alternatives, and refine solutions without the scheduling, alignment, or role-boundary constraints of a multi-person team.

At the same time, several new forms of coordination emerge:

- **Verification coordination**: ensuring that AI-generated code is correct, secure, maintainable, and aligned with architectural constraints; this often becomes the dominant cost.
- **Cognitive boundary management**: maintaining situational awareness despite rapid model-driven changes, a known challenge in human–AI teaming (Seeber, 2020; Schmutz, 2024).
- **Governance integration**: embedding automated tests, policy checks, and security analysis into AI-driven workflows.
- **Cross-unit alignment**: coordinating with other teams, systems, and organizational structures that still operate within human-centric rhythms and constraints.

These challenges imply that while AI may collapse coordination costs within a centaur unit, it does not eliminate the need for coordination across units. Architectures, organizational boundaries, compliance frameworks, and strategic direction all continue to impose constraints. Delivery frameworks and organizational models—designed for human-only teams—must therefore be re-evaluated in light of this altered distribution of coordination effort.

IMPLICATIONS FOR DELIVERY FRAMEWORKS AND ORGANIZATIONAL MODELS

The shift in the effective unit of work from a human team to a centaur unit has direct implications for how coordination structures add value. Frameworks that assume synchronous team rituals, strict role separations, or batch-oriented work may impose unnecessary overhead when a single centaur unit can execute the full delivery cycle. Conversely, organizational models that excel at managing cross-boundary coordination, governance, and architectural integrity may become more important, not less.

These considerations motivate the evaluation dimensions introduced in subsequent sections. Delivery frameworks are assessed along axes such as dependency on role specialization, synchronous coordination, and alignment with rapid exploration. Organizational models are assessed according to their ability to support decentralized decision-making, cross-boundary coherence, and robust governance in environments where centaur units operate as semi-autonomous micro-teams. The centaur development reality does not invalidate existing methods, but it shifts the criteria by which their suitability must be judged.

COORDINATION THEORY BASIS

Coordination has long been recognized as the central challenge of software engineering. Brooks (Brooks, 1995) identified communication overhead, conceptual integrity, and the difficulty of aligning multiple humans' mental models as dominant sources of cost. Conway's law (Conway, 1968) formalized the structural coupling between organizational communication patterns and software architectures, implying that coordination structures shape technical outcomes as much as technical decisions do.

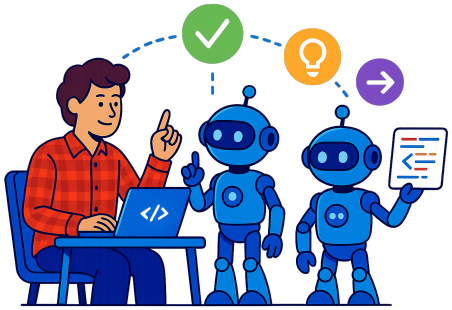
Organizational theory provides further grounding for these observations. Galbraith's information-processing model (Galbraith, 1974) frames organizations as mechanisms for managing uncertainty through coordination, specialization, and integration. Mintzberg's structural configurations (Mintzberg, 1979) describe how different organizational forms—functional, divisional, matrix, or adhocratic—balance centralization, autonomy, and information flow. Burns and Stalker's distinction between mechanistic and organic systems (Burns and Stalker, 1961) highlights how turbulent environments favor decentralized, adaptive structures over rigid hierarchies. Collectively, these bodies of work converge on a shared view: coordination is the primary limiting factor in complex, interdependent work.

Software delivery frameworks encode coordination strategies derived from these principles. Agile methods emphasize synchronous communication, shared responsibility, and cross-functional teams to reduce misalignment costs (Cockburn, 2001; Highsmith, 2001). Scrum operationalizes coordination through timeboxed events and role distinctions (Schwaber and Sutherland, 2017). Extreme Programming promotes tight feedback loops and collective code ownership (Beck, 1999). Kanban (Anderson, 2010) and Lean perspectives focus on flow efficiency and reduction of batch sizes. DevOps and SRE extend these ideas by automating integration, deployment, and monitoring to reduce coordination friction across the development–operations boundary (Forsgren, Humble, and Kim, 2018; Humble, Molesky, and O'Reilly, 2015). Each framework implicitly assumes that coordination occurs *between* humans, and that the difficulty of achieving shared context is a fundamental constraint.

Human–AI teaming research reinforces the importance of coordination mechanisms. Seeber et al. (Seeber, 2020) describe coordination, trust calibration, and responsibility attribution as recurring challenges when AI systems act as collaborators rather than tools. Schmutz et al. (Schmutz, 2024) emphasize the need for clear interaction patterns and governance structures to support mixed human–AI teams. Human-centered AI work (Shneiderman, 2020) similarly stresses oversight, transparency, and predictable behavior as prerequisites for safe and effective collaboration.

The post-2025 centaur development reality alters the distribution of coordination effort. In a human-only team, coordination costs arise primarily from communication, synchronization, and the maintenance of shared understanding. In a human–AI unit, many of these costs diminish: the engineer and AI agents can iterate asynchronously at high frequency, maintain a shared context through persistent state and prompts, and explore alternatives without incurring interpersonal alignment overhead. However, new forms of coordination emerge, including verification of AI-generated artifacts, governance of tool behavior, and alignment across organizational boundaries.

This shift does not eliminate the need for coordination; it redistributes it. Frameworks that rely on synchronous rituals, rigid role separations, or batch-oriented planning may impose unnecessary overhead when a single centaur unit can execute substantial portions of the delivery cycle internally. Conversely, organizational models that emphasize decentralized decision-making, modular interaction surfaces, and automated governance may better accommodate this altered landscape. These theoretical considerations motivate the evaluation dimensions introduced in the following sections, which assess delivery frameworks and organizational models according to their alignment with the coordination demands of centaur-based software development.



DELIVERY FRAMEWORK EVALUATION MODEL

To evaluate delivery frameworks under the centaur development paradigm, we require a scoring model that is transparent, reproducible, and aligned with the coordination theory foundations established earlier. This section defines six dimensions (D1–D6), each operationalized through three binary indicators. All indicators are phrased in positive form: *a score of 1 means alignment with centaur development, and a score of 0 means misalignment*. This ensures semantic consistency across dimensions and avoids interpretive inversion.

For any framework F, each indicator $I_{k,j}(F)$ takes the value 1 (aligned) or 0 (not aligned). The raw dimension score is:

$$R_k(F) = \tfrac{1}{3}\sum_{j=1}^{3} I_{k,j}(F), \qquad T_k(F) = \begin{cases} 0 & \text{if } R_k(F) < \tfrac{1}{3}, \\ 0.5 & \text{if } \tfrac{1}{3} \le R_k(F) < \tfrac{2}{3}, \\ 1 & \text{if } R_k(F) \ge \tfrac{2}{3}. \end{cases}$$

The resulting ternary vector $T(F)$ for each framework is thus mechanically derived and fully auditable. An overall compatibility score for a framework can be reported as the mean of its six ternary dimension scores:

$$T_{\text{overall}}(F) = \tfrac{1}{6}\sum_{k=1}^{6} T_k(F).$$

A quick pseudo-code summary (each indicator is 0 or 1):

```
i1, i2, i3 = three indicators for dimension
R = (i1 + i2 + i3) / 3
if R < 1/3:    T = 0
elif R < 2/3:  T = 0.5
else:          T = 1
```

Table 1: *Delivery Evaluation Dimensions (D1–D6): Summary*

Dimension	Focus
D1	Independence from synchronous team rituals
D2	Flexibility of roles and boundaries
D3	Support for rapid exploration and micro-iterations
D4	Alignment with automated governance and verification
D5	Architectural flexibility under uncertainty
D6	Compatibility with AI-agent workflows

D1: INDEPENDENCE FROM SYNCHRONOUS TEAM RITUALS

Centaur development reduces the need for synchronous human coordination by allowing rapid, asynchronous cycles between the engineer and AI agents. Heavy reliance on ceremonies imposes avoidable coordination cost. Based on coordination research (Cockburn, 2001; Schwaber and Sutherland, 2017), we evaluate:

D1.1) *Can the framework be executed without mandatory synchronous ceremonies?*
D1.2) *Does the framework emphasize artifacts, tooling, or asynchronous workflows rather than meetings as primary coordination mechanisms?*
D1.3) *Does the framework permit continuous progress without waiting for scheduled events?*

A high score indicates a lightweight, asynchronous coordination model compatible with centaur workflows.

D2: FLEXIBILITY OF ROLES AND BOUNDARIES

In centaur development, a single senior engineer spans many responsibilities. Rigid role separation can introduce unnecessary friction. Following organizational theory on role flexibility (Burns and Stalker, 1961; Mintzberg, 1979), we evaluate:

D2.1) *Does the framework allow roles to be combined or minimized?*
D2.2) *Are responsibilities described in capability terms rather than strict role boundaries?*
D2.3) *Can a single unit (human–AI) reasonably execute the method without requiring multiple human roles?*

High D2 indicates compatibility with generalist, micro-unit centaur teams.

D3: SUPPORT FOR RAPID EXPLORATION AND MICRO-ITERATIONS

Centaur development enables extremely rapid exploration. Frameworks that enforce rigid batch cycles or planning horizons impede this. Research on iteration speed (Forsgren, Humble, and Kim, 2018; Ries, 2011) motivates:

D3.1) *Does the framework allow continuous or near-continuous iteration?*
D3.2) *Can work items be re-planned or adapted at any time?*
D3.3) *Does the framework support micro-iterations (very small batch sizes) over fixed-length cycles?*

High scores indicate strong alignment with AI-accelerated iteration.

D4: ALIGNMENT WITH AUTOMATED GOVERNANCE AND VERIFICATION

Human–AI teaming increases the importance of automated verification and continuous governance (Forsgren, Humble, and Kim, 2018; Shneiderman, 2020). We evaluate:

D4.1) *Does the framework encourage automated testing or continuous integration as integral practices?*
D4.2) *Are fast feedback loops (tests, metrics, monitoring) structurally supported?*
D4.3) *Can governance be embedded into automated pipelines rather than relying on manual gates?*

High D4 indicates strong fit with AI-driven workflows requiring continuous assurance.

D5: ARCHITECTURAL FLEXIBILITY UNDER UNCERTAINTY

AI-assisted development reduces the cost of exploring design alternatives, making emergent architecture more viable (Sanchez and Mahoney, 1996; Larman and Vodde, 2017). We evaluate:

D5.1) *Does the framework support incremental or emergent architecture?*
D5.2) *Are refactoring and architectural evolution treated as routine rather than exceptional?*
D5.3) *Does the framework assume modularity or other structures enabling architectural adaptability?*

High D5 reflects compatibility with dynamic architectural evolution.

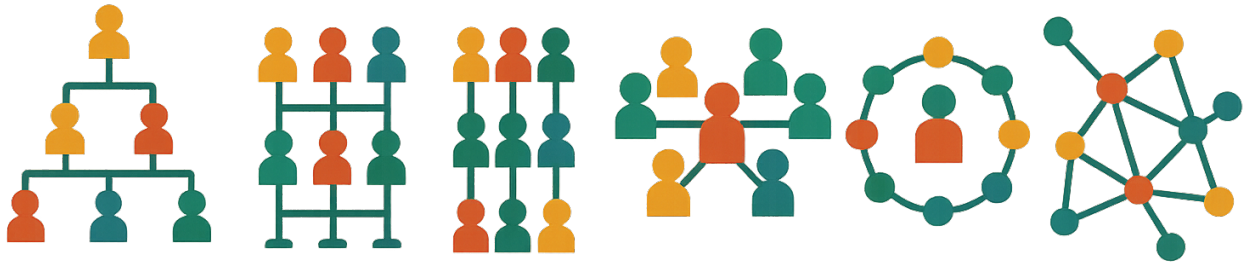
D6: COMPATIBILITY WITH AI-AGENT WORKFLOWS

Frameworks vary in how easily they incorporate automation, tool integration, and agentic execution. Based on socio-technical principles (Humble, Molesky, and O'Reilly, 2015; Ries, 2011), we evaluate:

D6.1) *Is the framework agnostic to whether execution is performed by humans, tools, or AI agents?*
D6.2) *Does the framework encourage extensive automation and integration with tooling?*
D6.3) *Would the framework's coordination mechanisms remain valid if a large proportion of execution were automated?*

High D6 indicates conceptual synergy with agentic development.

Together, these six dimensions form a consistent, positive-indicator evaluation model. They allow delivery frameworks to be assessed transparently with respect to the coordination demands of centaur development. Evaluation of Delivery Frameworks (p. 14) applies this model to a set of widely used frameworks.



ORGANIZATIONAL MODEL EVALUATION MODEL

Organizational structures determine how coordination, authority, and information flow scale across socio-technical systems. Under centaur development, intra-unit coordination becomes inexpensive, while inter-unit coordination, governance, and architectural alignment remain essential. Consequently, organizational models must be evaluated for their ability to support semi-autonomous centaur micro-units while still enabling coherence across a larger system.

This section defines six organizational dimensions (O1–O6), each evaluated using three binary indicators phrased in *positive form*: a value of 1 denotes alignment with centaur development, while 0 denotes misalignment. This ensures consistency with the delivery-model scoring from Delivery Framework Evaluation Model (p. 8).

For any organizational model M:

$$R_k(M) = \tfrac{1}{3}\sum_{j=1}^{3} I_{k,j}(M), \qquad T_k(M) = \begin{cases} 0 & \text{if } R_k(M) < \tfrac{1}{3}, \\ 0.5 & \text{if } \tfrac{1}{3} \le R_k(M) < \tfrac{2}{3}, \\ 1 & \text{if } R_k(M) \ge \tfrac{2}{3}. \end{cases}$$

A quick pseudo-code summary (each indicator is 0 or 1):

```
i1, i2, i3 = three indicators for dimension
R = (i1 + i2 + i3) / 3
if R < 1/3:    T = 0
elif R < 2/3:  T = 0.5
else:          T = 1
```

A high score indicates strong compatibility with the coordination needs of centaur development. An overall compatibility score for a model can be reported as the mean of its six ternary dimension scores:

$$T_{\text{overall}}(M) = \tfrac{1}{6}\sum_{k=1}^{6} T_k(M).$$

Table 2: *Organizational Evaluation Dimensions (O1–O6): Summary*

Dimension	Focus
O1	Local autonomy and span-of-control flexibility
O2	Decentralization of decision authority
O3	Efficiency of cross-boundary alignment mechanisms
O4	Integration of automated governance and compliance
O5	Adaptability to high-flux, high-iteration work
O6	Accommodation of micro-units and variable topology

O1: LOCAL AUTONOMY AND SPAN-OF-CONTROL FLEXIBILITY

Organizations with broad local decision rights and limited hierarchical interference enable centaur units to iterate quickly and adaptively, consistent with organic structures (Burns and Stalker, 1961; Mintzberg, 1979). We evaluate:

O1.1) *Can delivery units make routine decisions without requiring multi-layer hierarchical approval?*
O1.2) *Is the span of control structured such that supervisors oversee relatively few units, allowing rapid vertical communication when needed?*
O1.3) *Are units explicitly empowered to select or adapt their own workflows, processes, or engineering approaches?*

High O1 indicates an organizational environment that does not impede the autonomy of centaur micro-units.

O2: DECENTRALIZATION OF DECISION AUTHORITY

Galbraith's information-processing view (Galbraith, 1974) and agile organizational principles (Highsmith, 2001) emphasize that decisions should be made where information resides. For centaur development, this is the human–AI unit. We evaluate:

O2.1) *Are prioritization, architectural, or implementation decisions delegated to the teams or units closest to the work?*
O2.2) *Does the model minimize reliance on PMOs, steering committees, or central boards for tactical decisions?*
O2.3) *Is decentralized decision-making explicitly encouraged in high-uncertainty or rapidly changing environments?*

High O2 indicates strong alignment with distributed, knowledge-based decision authority required by centaur workflows.

O3: EFFICIENCY OF CROSS-BOUNDARY ALIGNMENT MECHANISMS

Centaur units reduce coordination within a team, but cross-unit alignment remains essential. Modular systems theory (Sanchez and Mahoney, 1996) and team-based topologies (Skelton and Pais, 2019) emphasize minimizing synchronous cross-team coordination through clear boundaries and interfaces. We evaluate:

O3.1) *Does the model emphasize modular boundaries (APIs, SLAs, service contracts) over meetings for cross-team alignment?*
O3.2) *Are dependencies managed primarily through artifacts or tooling rather than synchronous coordination sessions?*
O3.3) *Does the model support platformization or boundary-setting structures that reduce horizontal coordination load?*

High O3 reflects an organization that scales effectively as centaur units multiply.

O4: INTEGRATION OF AUTOMATED GOVERNANCE AND COMPLIANCE

Human–AI teaming requires continuous verification, policy enforcement, and risk management (Shneiderman, 2020; Schmutz, 2024). Organizational models built on manual, centralized approval processes impede AI-accelerated iteration. We evaluate:

O4.1) *Does the model encourage automation of governance (e.g., policy-as-code, automated compliance checks)?*
O4.2) *Are risk and quality controls embedded within delivery units rather than externalized to committees or gatekeepers?*
O4.3) *Does the model support continuous, non-blocking governance aligned with frequent releases?*

High O4 indicates strong alignment with continuous, AI-compatible governance.

O5: ADAPTABILITY TO HIGH-FLUX, HIGH-ITERATION WORK

Centaur development increases the rate at which work evolves. Rigid planning cycles slow adaptation (Ries, 2011; Forsgren, Humble, and Kim, 2018). We evaluate:

O5.1) *Does the model support flexible or continuous planning rather than fixed quarterly or annual cycles?*
O5.2) *Can priorities be adjusted rapidly based on new information?*
O5.3) *Does the model reduce batching of decisions, approvals, and coordination events?*

High O5 indicates strong compatibility with AI-accelerated iteration.

O6: ACCOMMODATION OF MICRO-UNITS AND VARIABLE TOPOLOGY

Classical management models assume multi-person teams (Schwaber and Sutherland, 2017; Mintzberg, 1979), but centaur development often operates at micro-unit scale (1–3 engineers). Modern modular structures (Sanchez and Mahoney, 1996; Skelton and Pais, 2019) support variable boundaries. We evaluate:

O6.1) *Does the model recognize 1–3 person units as legitimate structural entities?*
O6.2) *Does it avoid mandating fixed team sizes or cross-functional compositions?*
O6.3) *Does the model support dynamic topologies—units merging, splitting, or embedding—without structural penalty?*

High O6 indicates strong structural fit with centaur micro-units.

Together, these six dimensions form a coherent, indicator-based model for evaluating organizational structures under centaur development. They parallel the delivery framework dimensions (D1–D6) while shifting the emphasis to authority distribution, boundary management, and governance integration in AI-accelerated environments. Evaluation of Organizational Models (p. 25) applies this model to a representative set of organizational structures.

EVALUATION OF DELIVERY FRAMEWORKS

This section evaluates a representative set of delivery frameworks using the indicator-based scoring model from Delivery Framework Evaluation Model (p. 8). All indicators are positively phrased, where a score of 1 indicates alignment with centaur development and 0 indicates misalignment. Raw scores $R_k(F)$ and ternary compatibility scores $T_k(F)$ are computed mechanically according to the rules defined earlier.

Each subsection provides:

(i) a brief rationale,
(ii) a 0/1 indicator table with justification,
(iii) raw and ternary dimension scores.

SCRUM

Scrum (Schwaber and Sutherland, 2017) assumes synchronous coordination, distinct roles, and fixed-length Sprints. These assumptions reflect a human-centric coordination model.

ID	Val	Rationale
D1.1	0	Scrum cannot operate without mandatory Sprint ceremonies.
D1.2	0	Coordination relies on events rather than asynchronous mechanisms.
D1.3	0	Progress depends on scheduled meetings, not continuous flow.
D2.1	0	Roles (PO, SM, Developers) are mandatory and non-collapsible.
D2.2	0	Responsibilities are rigidly partitioned.
D2.3	0	A centaur unit cannot satisfy required multi-role structure.
D3.1	0	Re-planning cannot occur freely within Sprints.
D3.2	0	Iteration length is fixed.
D3.3	0	Sprint batch size is not tunable.
D4.1	0	Scrum does not prescribe CI/CD or automation.
D4.2	1	Potentially shippable increments allow automated inspection if adopted.
D4.3	0	Governance relies on team events, not automated gates.
D5.1	0	Scrum is architecture-neutral; does not encourage emergent design.
D5.2	1	Refactoring is common practice within Scrum teams.
D5.3	0	No structural focus on modularity.
D6.1	0	Scrum assumes manual, human execution.
D6.2	0	Automation is optional and external to the framework.
D6.3	0	AI-agent workflows conflict with ceremony-driven coordination.

INDICATOR TABLE.

SCORES.

$$R = \{0, 0, 0, 1/3, 1/3, 0\}, \qquad T = \{0, 0, 0, 0.5, 0.5, 0\}.$$

EXTREME PROGRAMMING (XP)

XP (Beck, 1999) emphasizes technical excellence, automation, and continuous feedback. Some synchronous practices limit full centaur alignment, but XP supports micro-iterations and verification exceptionally well.

ID	Val	Rationale
D1.1	0	Pair programming is synchronous and central.
D1.2	1	XP's coordination is primarily artifact-based (tests, CI), not ceremony-driven.
D1.3	1	Work continues asynchronously without ceremony gates.
D2.1	1	Roles are flexible; developers share responsibilities.
D2.2	1	Capability-based, collective ownership.
D2.3	0	XP assumes at least small multi-person teams; a single centaur unit cannot fully enact pair programming.
D3.1	1	Continuous re-planning and test-first iterations.
D3.2	1	Micro-iterations are intrinsic.
D3.3	1	Batch sizes are minimal and tunable.
D4.1	1	TDD mandates automated tests.
D4.2	1	CI and continuous feedback are core.
D4.3	1	Governance emerges from automated tests and pipelines.
D5.1	1	Emergent architecture is explicitly encouraged.
D5.2	1	Continuous refactoring is central.
D5.3	1	Simple design promotes modularity.
D6.1	0	XP predates AI and assumes human execution.
D6.2	1	Strong tooling/automation culture.
D6.3	0	Some practices (pair programming) assume human participation.

INDICATOR TABLE.

SCORES.

$$R = \{2/3,\ 2/3,\ 1,\ 1,\ 1,\ 1/3\}, \qquad T = \{0.5,\ 1,\ 1,\ 1,\ 1,\ 0.5\}.$$

KANBAN

Kanban (Anderson, 2010) is lightweight, asynchronous, and oriented toward continuous flow. Lack of prescriptive structure makes it flexible but leaves governance and architecture external.

ID	Val	Rationale
D1.1	1	No mandatory ceremonies.
D1.2	1	Coordination is visual and asynchronous.
D1.3	1	Work flows continuously without scheduled gates.
D2.1	1	No required roles; highly flexible.
D2.2	1	Responsibilities are capability-based.
D2.3	1	A single unit can operate Kanban independently.
D3.1	1	Continuous flow is intrinsic.
D3.2	1	Re-planning occurs at any time.
D3.3	1	Work item sizes are tunable.
D4.1	0	Automation is not defined in the method.
D4.2	1	Feedback loops can be embedded.
D4.3	0	Governance is not structurally embedded.
D5.1	0	Architecture is external to Kanban.
D5.2	1	Continuous refactoring is compatible.
D5.3	0	No explicit modularity support.
D6.1	1	Kanban is execution-agnostic.
D6.2	1	Automation integrates easily.
D6.3	1	Its coordination model still works if execution is automated.

INDICATOR TABLE.

SCORES.

$$R = \{1,\ 1,\ 1,\ 1/3,\ 1/3,\ 1\}, \qquad T = \{1,\ 1,\ 1,\ 0.5,\ 0.5,\ 1\}.$$

LEAN STARTUP

Lean Startup (Ries, 2011) is conceptually among the closest matches to centaur development due to high emphasis on experimentation, telemetry, and rapid feedback.

ID	Val	Rationale
D1.1	1	No required synchronous events.
D1.2	1	Coordination is artifact- and metric-based.
D1.3	1	Continuous iteration is natural.
D2.1	1	No prescribed roles.
D2.2	1	Responsibilities defined through capabilities.
D2.3	1	Fully operable by a centaur unit.
D3.1	1	Continuous iteration is core.
D3.2	1	Re-planning is continuous.
D3.3	1	Micro-iterations are encouraged.
D4.1	1	Emphasizes automated measurement and learning.
D4.2	1	Feedback loops define the method.
D4.3	1	Governance-by-metrics aligns with centaur workflows.
D5.1	1	Supports emergent architecture through experimentation.
D5.2	1	Encourages revisiting assumptions continuously.
D5.3	1	Prefers modular, adaptive systems.
D6.1	1	Execution mode is irrelevant.
D6.2	1	Automation enhances learning loops.
D6.3	1	Structure remains valid under AI execution.

INDICATOR TABLE.

SCORES.

$$R = \{1, 1, 1, 1, 1, 1\}, \qquad T = \{1, 1, 1, 1, 1, 1\}.$$

DEVOPS / SRE

DevOps and SRE emphasize automation, continuous verification, and operational excellence (Forsgren, Humble, and Kim, 2018; Humble, Molesky, and O'Reilly, 2015). They naturally align with centaur workflows.

ID	Val	Rationale
D1.1	1	No mandatory ceremonies.
D1.2	1	Coordination through tooling, pipelines, and CI/CD.
D1.3	1	Continuous progress is the norm.
D2.1	1	Roles are flexible.
D2.2	1	Focus is on capabilities (dev + ops).
D2.3	1	A centaur unit can operate DevOps workflows.
D3.1	1	Continuous iteration is intrinsic.
D3.2	1	Re-planning happens continuously.
D3.3	1	Micro-iterations supported via automation.
D4.1	1	Automation central.
D4.2	1	CI/CD, monitoring, and metrics built-in.
D4.3	1	Governance integrated into pipelines.
D5.1	1	Encourages evolutionary architecture.
D5.2	1	Emphasizes loosely coupled systems.
D5.3	1	Supports modularity.
D6.1	1	Fully execution-agnostic.
D6.2	1	AI agents can easily integrate.
D6.3	1	Coordination remains coherent under automation.

INDICATOR TABLE.

SCORES.

$$R = \{1, 1, 1, 1, 1, 1\}, \qquad T = \{1, 1, 1, 1, 1, 1\}.$$

SAFE

SAFe (Scaled Agile Inc., 2021) is heavy on roles, ceremonies, and centralized governance. It is the least compatible scaled framework.

ID	Val	Rationale
D1.1	0	Mandatory PI Planning and ART events.
D1.2	0	Heavy synchronous coordination.
D1.3	0	Cannot operate continuously without ceremonies.
D2.1	0	Extensive role hierarchy.
D2.2	0	Strict responsibility separation.
D2.3	0	Centaur unit cannot fulfill role architecture.
D3.1	0	Fixed PI cadence.
D3.2	0	Re-planning is gated.
D3.3	0	Batch size inflexible.
D4.1	0	Governance is manual/stage-gated.
D4.2	0	Automated feedback optional, not structural.
D4.3	0	Governance boards act as gates.
D5.1	1	Architectural guidance exists.
D5.2	0	Refactoring constrained by planning.
D5.3	0	Architecture evolves slowly.
D6.1	0	Assumes human-centric execution.
D6.2	0	Automation is add-on.
D6.3	0	Agentic workflows conflict with structure.

INDICATOR TABLE.

SCORES.

$$R = \{0, 0, 0, 0, 1/3, 0\}, \qquad T = \{0, 0, 0, 0, 0.5, 0\}.$$

LESS

LeSS (Larman and Vodde, 2017) reduces hierarchical elements but retains synchronous coordination and Sprint cadences.

ID	Val	Rationale
D1.1	0	Sprint events required.
D1.2	0	Coordination is synchronous.
D1.3	1	Some continuous flow allowed between Sprints.
D2.1	1	Fewer roles than SAFe.
D2.2	0	Some role rigidity persists.
D2.3	0	Centaur unit cannot meet role expectations.
D3.1	0	Sprint cadence fixed.
D3.2	1	Re-planning possible between Sprints.
D3.3	0	Batch size tied to Sprint length.
D4.1	0	Automation not defined structurally.
D4.2	1	Feedback loops possible.
D4.3	0	Governance not embedded.
D5.1	1	Encourages emergent architecture.
D5.2	1	Refactoring supported.
D5.3	1	Modular design encouraged.
D6.1	0	Human-centric execution assumed.
D6.2	0	Automation externalized.
D6.3	0	Agentic workflows not supported.

INDICATOR TABLE.

SCORES.

$$R = \{1/3,\ 1/3,\ 1/3,\ 1/3,\ 1,\ 0\}, \qquad T = \{0.5,\ 0.5,\ 0.5,\ 0.5,\ 1,\ 0\}.$$

WATERFALL / V-MODEL

Plan-driven methods such as Waterfall and V-Model (Royce, 1970) assume sequential phases, upfront architecture, formal handoffs, and manual governance. These assumptions conflict with centaur-style rapid iteration and automation.

ID	Val	Rationale
D1.1	0	Waterfall depends on milestone reviews and formal meetings.
D1.2	0	Coordination is document- and meeting-driven, not asynchronous.
D1.3	0	Progress halts at phase gates; continuous operation is impossible.
D2.1	0	Rigid role separation (analyst, designer, developer, tester).
D2.2	0	Responsibilities are strictly non-overlapping.
D2.3	0	Centaur micro-units cannot satisfy role partitioning.
D3.1	0	No re-planning between phases.
D3.2	0	Rapid iteration is not permitted structurally.
D3.3	0	Batch sizes (phases) are fixed and large.
D4.1	0	Governance is manual and stage-gated.
D4.2	0	Feedback loops occur only at phase boundaries.
D4.3	0	No support for automated quality gates.
D5.1	0	Assumes upfront architecture.
D5.2	0	Refactoring is discouraged or costly.
D5.3	0	No modularity assumptions; architecture is monolithic.
D6.1	0	Execution is fully human and sequential.
D6.2	0	Automation is not structurally recognized.
D6.3	0	Agentic workflows directly contradict the method.

INDICATOR TABLE.

SCORES.

$$R = \{0, 0, 0, 0, 0, 0\}, \qquad T = \{0, 0, 0, 0, 0, 0\}.$$

PRINCE2

PRINCE2 (Office of Government Commerce, 2009) is a project governance methodology optimized for predictability, control, and sequential stage boundaries. It is structurally incompatible with centaur-style high-frequency iteration.

ID	Val	Rationale
D1.1	0	Mandatory stage boundary reviews and governance meetings.
D1.2	0	Coordination is synchronous and document-driven.
D1.3	0	Continuous flow is impossible under stage boundaries.
D2.1	0	Rigid role structure (Project Board, PM, Team Manager).
D2.2	0	Responsibilities are strictly partitioned.
D2.3	0	Centaur units cannot satisfy mandatory governance roles.
D3.1	0	Iteration does not exist; waterfall planning dominates.
D3.2	0	Re-planning only allowed at stage boundaries.
D3.3	0	Batch size (stage) fixed and large.
D4.1	0	Governance is manual and gate-based.
D4.2	0	Automated feedback loops not part of the method.
D4.3	0	Compliance enforced via reviews, not pipelines.
D5.1	0	Expects upfront architecture or design stage.
D5.2	0	Architecture evolution conflicts with sequential control.
D5.3	0	No emphasis on modularity or adaptability.
D6.1	0	Assumes sequential human execution.
D6.2	0	Not designed for automation or tooling integration.
D6.3	0	Agentic workflows break the governance model entirely.

INDICATOR TABLE.

SCORES.

$$R = \{0, 0, 0, 0, 0, 0\}, \qquad T = \{0, 0, 0, 0, 0, 0\}.$$

OBAF

OBAF (Blomgren, 2025b) emphasizes outcome alignment, rapid experimentation, and minimal ceremony, making it structurally compatible with centaur-style development. This evaluation treats OBAF neutrally–as one example of a lightweight, outcome-oriented method.

ID	Val	Rationale
D1.1	1	No mandatory synchronous events.
D1.2	1	Coordination is artifact- and outcome-driven.
D1.3	1	Continuous progress is natural.
D2.1	1	No required roles; fully flexible.
D2.2	1	Responsibilities defined through outcomes and metrics.
D2.3	1	A single centaur unit can execute OBAF end-to-end.
D3.1	1	Designed for continuous iteration.
D3.2	1	Re-planning occurs in real time based on feedback.
D3.3	1	Small, frequent learning cycles encouraged.
D4.1	1	Strong integration of automated metrics and telemetry.
D4.2	1	Feedback loops are at the core.
D4.3	1	Governance designed around metric-based, automated alignment.
D5.1	1	Encourages evolutionary architecture.
D5.2	1	Refactoring and adaptation are routine.
D5.3	1	Supports modular and exploratory design.
D6.1	1	Execution-agnostic by design.
D6.2	1	Encourages automation and agentic workflows.
D6.3	1	Remains coherent even if execution is automated.

INDICATOR TABLE.

SCORES.

$$R = \{1, 1, 1, 1, 1, 1\}, \qquad T = \{1, 1, 1, 1, 1, 1\}.$$

SUMMARY OF DELIVERY FRAMEWORK COMPATIBILITY SCORES

Table 3 consolidates the ternary compatibility scores $T_k(F)$ for all evaluated delivery frameworks across the six dimensions D1–D6. These scores are not intended as rankings but as structured reflections of how each framework's coordination assumptions align with the centaur development characteristics described in The Post-2025 Centaur Development Reality (p. 4) and Delivery Framework Evaluation Model (p. 8). The values are derived mechanically from the binary indicators presented in the preceding subsections, making the scoring transparent and reproducible.

High scores (1) indicate strong conceptual and structural compatibility with centaur workflows, particularly in areas such as asynchronous coordination, role flexibility, micro-iteration support, automated verification, architectural adaptability, and AI-agent integration. Mid-level scores (0.5) reflect partial compatibility or context-dependent alignment. Scores of 0 indicate structural misalignment with one or more key coordination demands of centaur development.

The summary table allows direct comparison of frameworks across dimensions and provides a basis for the analysis in later sections, including the evaluation of organizational models and the case in Case Illustration (p. 39).

Table 3: *Delivery Framework Compatibility Scores (D1–D6)*

Framework	D1	D2	D3	D4	D5	D6
Scrum	0	0	0	0.5	0.5	0
XP	0.5	1	1	1	1	0.5
Kanban	1	1	1	0.5	0.5	1
Lean Startup	1	1	1	1	1	1
DevOps/SRE	1	1	1	1	1	1
SAFe	0	0	0	0	0.5	0
LeSS	0.5	0.5	0.5	0.5	1	0
Waterfall/V-Model	0	0	0	0	0	0
PRINCE2	0	0	0	0	0	0
OBAF	1	1	1	1	1	1

EVALUATION OF ORGANIZATIONAL MODELS

This section evaluates twelve organizational structures using the indicator-based scoring model developed in Organizational Model Evaluation Model (p. 11). These structural forms represent widely adopted models in contemporary organizations as well as influential alternatives proposed in organizational theory. They vary significantly in their assumptions about decision rights, information flow, team topology, governance, and coordination mechanisms. Such assumptions become critical to examine under the centaur development paradigm, where a human–AI micro-unit can perform end-to-end work with substantially reduced intra-unit coordination cost but where inter-unit alignment, governance, and architectural coherence remain essential.

Each organizational model M is evaluated along the six dimensions O1–O6, covering autonomy, decision decentralization, cross-boundary efficiency, governance integration, adaptability to high-flux iteration, and support for micro-unit structures. Each dimension consists of three binary indicators $I_{k,j}(M)$, where 1 denotes alignment with centaur development and 0 denotes misalignment. Raw scores are computed as:

$$R_k(M) = \frac{1}{3} \sum_{j=1}^{3} I_{k,j}(M),$$

and mapped to the ternary compatibility scale:

$$T_k(M) = \begin{cases} 0 & \text{if } R_k(M) < \frac{1}{3}, \\ 0.5 & \text{if } \frac{1}{3} \leq R_k(M) < \frac{2}{3}, \\ 1 & \text{if } R_k(M) \geq \frac{2}{3}. \end{cases}$$

A high ternary score indicates that an organizational model provides the structural conditions—autonomy, local decision authority, asynchronous coordination, embedded governance, and flexible boundary management—required for centaur-style software delivery. A low score indicates structural tension or misalignment with these coordination needs.

The following subsections evaluate each organizational model individually. The first model, the *Functional Hierarchy*, represents the baseline against which more modern or decentralized structures can be compared.

FUNCTIONAL HIERARCHY

The functional hierarchy is a classical organizational form in which work is divided according to professional specialization (e.g., frontend engineering, backend engineering, QA, product management, operations). Although this design optimizes for efficiency and depth of expertise, it creates strong dependencies across functions and relies heavily on synchronous coordination and managerial escalation. Such properties make it structurally misaligned with the autonomy, local decision authority, and asynchronous coordination patterns characteristic of centaur development.

ID	Val	Rationale
O1.1	0	Local teams cannot make routine decisions without escalation to functional managers.
O1.2	0	Managers often oversee large spans across specialized units, slowing vertical communication.
O1.3	0	Workflows and processes are standardized per function; units cannot autonomously adapt them.
O2.1	0	Architecture, prioritization, and technical decisions are centralized in management.
O2.2	0	PMOs or central committees frequently control tactical decision-making.
O2.3	0	Model does not support decentralized decision-making in high-uncertainty environments.
O3.1	0	Cross-functional dependencies require meetings, handoffs, and escalations.
O3.2	0	Interfaces between functions rely on synchronous communication rather than artifacts.
O3.3	0	No structural support for modular or contract-based alignment.
O4.1	0	Governance is manual and committee-driven.
O4.2	0	Quality, security, and risk checks are handled by specialized functions, not within teams.
O4.3	0	Continuous or automated governance is not structurally enabled.
O5.1	0	Planning cycles are long (quarterly or annual).
O5.2	0	Priority changes require negotiation across functions, slowing responsiveness.
O5.3	0	Decision batching is inherent to functional planning.
O6.1	0	Structure assumes multi-person teams, not micro-units.
O6.2	0	Team size and roles are predetermined by function.
O6.3	0	Merging or splitting micro-units is structurally impossible.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}, \qquad T = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}.$$

The functional hierarchy shows complete misalignment with centaur development. Its reliance on specialization, centralization, synchronous communication, and slow planning cycles conflicts with the autonomy, modularity, and rapid iteration characteristics of human–AI micro-units.

DIVISIONAL (M-FORM) ORGANIZATION

The divisional or multidivisional (M-form) organization structures the company into semi-autonomous business units organized around product lines, geographic markets, or customer segments. Each division typically maintains internal functional departments (e.g., engineering, marketing, finance) and reports to a central corporate headquarters that sets strategic direction and allocates resources. While M-form structures increase accountability and local ownership relative to a pure functional hierarchy, they still rely heavily on centralized governance, multi-layered decision processes, and synchronous alignment mechanisms. These properties limit compatibility with centaur-style micro-unit execution, where autonomy, rapid iteration, and lightweight cross-boundary coordination are essential.

ID	Val	Rationale
O1.1	0	Divisions retain approval chains for routine decisions; autonomy is limited by divisional leadership.
O1.2	0	Multiple managerial layers within each division constrain rapid vertical communication.
O1.3	0	Teams within divisions must follow standardized divisional operating models and cannot freely adapt workflows.
O2.1	0	Key decisions (architecture, prioritization, investment) are owned by divisional management, not delivery units.
O2.2	0	Divisions rely on central corporate functions (finance, strategy, architecture) for major decisions.
O2.3	0	The model is built around hierarchical decision-making and does not encourage local autonomy in uncertainty.
O3.1	0	Cross-division coordination requires steering groups or escalation pathways.
O3.2	0	Dependencies are managed via synchronous communication, budgets, and centralized processes.
O3.3	0	Divisional structures rarely implement modular, contract-based interaction surfaces.
O4.1	0	Governance is centralized and manual, driven by corporate review cycles.
O4.2	0	Risk and compliance functions are external to delivery units and impose oversight.
O4.3	0	Automated or continuous governance mechanisms are uncommon in M-form structures.
O5.1	0	Planning cycles are quarterly or annual, aligned with corporate budgeting.
O5.2	0	Priorities cannot shift rapidly; reprioritization must propagate through divisional layers.
O5.3	0	Strategic and operational decisions are batched at division- or portfolio-level cycles.
O6.1	0	Divisions assume teams of substantial size; micro-units are neither recognized nor supported.
O6.2	0	Team size and composition is strongly determined by divisional functional departments.
O6.3	0	Divisions are static and do not support dynamic merging or splitting of units.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}, \qquad T = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}.$$

Like the functional hierarchy, the divisional M-form organization is deeply misaligned with centaur development. Its hierarchical authority gradients, synchronous cross-boundary coordination, centralized governance, and reliance on multi-person functional teams inhibit the autonomy, rapid iteration, and modular interaction patterns that centaur micro-units require.

MATRIX ORGANIZATION

The matrix organization overlays two orthogonal structures—typically functional departments and cross-functional project or product groupings. Employees report to both a functional manager (who controls professional standards, resourcing, and career progression) and a project/product manager (who directs day-to-day work priorities). While the matrix was historically intended to improve cross-functional collaboration and resource flexibility, it is widely recognized as a coordination-heavy structure that imposes high cognitive load, dual authority conflicts, and frequent alignment meetings. These characteristics run counter to the autonomy, clarity of ownership, and rapid iteration cadence required by centaur development.

ID	Val	Rationale
O1.1	0	Matrix organizations require approvals from both functional and project managers; local autonomy is severely limited.
O1.2	0	Dual reporting increases managerial layers and slows vertical communication.
O1.3	0	Teams cannot autonomously alter workflows because functional and project structures impose conflicting constraints.
O2.1	0	Key decisions frequently require consensus between two authority lines, impeding decentralization.
O2.2	0	PMOs, portfolio boards, and functional committees mediate most tactical decisions.
O2.3	0	The model is explicitly designed around shared authority rather than local decision rights in dynamic contexts.
O3.1	0	Cross-boundary alignment occurs through extensive meetings involving multiple managers.
O3.2	0	Dependencies are handled via synchronous negotiation rather than artifact-centric coordination.
O3.3	0	Matrix structures do not naturally enforce modular boundaries; instead, they create overlapping responsibilities.
O4.1	0	Governance relies on functional reviews and project steering groups, not automation.
O4.2	0	Risk and quality functions are external and typically centralized within functional units.
O4.3	0	Continuous or embedded governance is structurally unsupported due to multiple supervisory layers.
O5.1	0	Planning cycles follow functional and project rhythms (usually quarterly or annual).
O5.2	0	Priorities cannot shift rapidly because changes must be negotiated across authority lines.
O5.3	0	Decision batching is intrinsic to matrix planning and resource allocation.
O6.1	0	Matrix structures assume medium-to-large project teams; they do not support micro-units.
O6.2	0	Team composition is determined jointly by functional managers and project leadership, eliminating size flexibility.
O6.3	0	Dynamic reconfiguration (merge/split) is constrained by dual reporting requirements.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}, \qquad T = \{0,\, 0,\, 0,\, 0,\, 0,\, 0\}.$$

The matrix organization scores a complete zero across all centaur-relevant dimensions. Dual authority structures, synchronous coordination, centralized governance, and rigidity of team composition make matrix organizations among the least compatible structures with centaur-based delivery. Coordination theory consistently identifies the matrix as one of the highest-overhead organizational forms, and centaur development amplifies these structural limitations.

CROSS-FUNCTIONAL PRODUCT TEAMS

Cross-functional product teams—popularized through Agile and Lean practices—are designed to bring all skills required for delivering customer value into a single, stable unit. These teams typically own a product or capability end-to-end, with responsibility spanning design, development, testing, and operations. While they represent a major improvement over functional or matrix structures, they still assume multi-person teams, synchronous rituals, and shared human decision-making. These assumptions align partially, but not fully, with the centaur development model, where a single senior engineer augmented by AI agents can often perform the work of an entire cross-functional team.

ID	Val	Rationale
O1.1	1	Teams typically have local authority over routine product and technical decisions.
O1.2	1	Flat team structures reduce hierarchical layers, enabling relatively fast communication.
O1.3	1	Teams often adapt their own ways of working (e.g., choosing Kanban or Scrum variants).
O2.1	1	Most decisions relevant to the product are made within the team; decision authority is localized.
O2.2	1	Cross-team or centralized governance typically handles portfolio-level issues, not day-to-day work.
O2.3	1	Agile product teams explicitly support decentralized decision-making.
O3.1	0	Cross-team dependencies still require coordination meetings, not modular interfaces.
O3.2	0	Artifacts (APIs, SLAs) vary widely; alignment often relies on human negotiation.
O3.3	0	Most organizations do not structure cross-team work using formal modularity or contract-based structures.
O4.1	0	Governance is usually manual (reviews, approvals) unless paired with DevOps–which is external to the team model.
O4.2	1	Teams often handle quality and risk controls internally (tests, reviews, acceptance criteria).
O4.3	0	Continuous automated governance is not an inherent structural property of the model.
O5.1	1	Teams typically follow flexible planning cadences (2–4 week cycles or continuous flow).
O5.2	1	Priorities can shift quickly through backlog management or product owner decisions.
O5.3	1	Little batching is mandatory within the team; iteration frequency is high.
O6.1	0	Model assumes teams of 5–9 people; micro-units are not supported as a structural form.
O6.2	0	Roles and team size expectations make 1–3 person teams non-standard.
O6.3	0	Teams are expected to be stable and long-lived; dynamic merging or splitting is discouraged.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 0,\ 1/3,\ 1,\ 0\} \quad \Rightarrow \quad T = \{1,\ 1,\ 0,\ 0.5,\ 1,\ 0\}.$$

Cross-functional product teams align strongly with centaur needs in autonomy (O1), decentralization (O2), and iteration speed (O5). However, they remain limited in three areas: cross-boundary modularity (O3), automated governance (O4), and especially micro-unit accommodation (O6). While these teams represent a significant improvement over functional or matrix structures, they are still anchored in the assumption that value is created by stable human teams rather than by human–AI micro-units.

TEAM TOPOLOGIES

Team Topologies (Skelton and Pais, 2019) proposes four fundamental team types (stream-aligned, enabling, complicated-subsystem, and platform teams) and three interaction modes (collaboration, X-as-a-Service, and facilitation). The model emphasizes reducing cognitive load, establishing clear ownership boundaries, promoting fast flow, and encouraging well-defined interaction surfaces across teams. These properties align well with the coordination demands of centaur development, especially in cross-boundary alignment (O3) and governance integration (O4). However, the model still assumes multi-person teams (rather than micro-units), which limits its alignment with O6.

ID	Val	Rationale
O1.1	1	Stream-aligned teams have authority over most day-to-day decisions within their domain.
O1.2	1	Team Topologies minimizes hierarchical layers and emphasizes clear team ownership.
O1.3	1	Teams are expected to evolve their own internal processes based on their cognitive load and workflow needs.
O2.1	1	Decision-making is intentionally pushed to the team boundaries; teams own their capabilities.
O2.2	1	Centralized oversight is minimized; platform and enabling teams support autonomy rather than control it.
O2.3	1	The model is explicitly designed for rapid decision-making in dynamic environments.
O3.1	1	Team boundaries are defined via service APIs, contracts, and platform interfaces, reducing coordination load.
O3.2	1	Interactions are formalized through X-as-a-Service or asynchronous collaboration modes.
O3.3	1	The model explicitly promotes modular, contract-driven interaction surfaces.
O4.1	1	Automation and platform capabilities are integral; governance can be embedded in service APIs.
O4.2	1	Risk and quality responsibilities are pushed into team workflows through strong ownership boundaries.
O4.3	1	Platform teams enable continuous, automated governance mechanisms across stream-aligned teams.
O5.1	1	The model supports continuous delivery and rapid evolution via reduced cognitive load.
O5.2	1	Teams are designed to adapt rapidly; flow metrics guide reprioritization.
O5.3	1	Decision batching is minimized through autonomous, stream-aligned ownership.
O6.1	0	Teams are explicitly assumed to be multi-person, often 5–9 people.
O6.2	0	Micro-unit configurations are not recognized as a core topology, though they are not forbidden.
O6.3	0	The model presumes relatively stable team boundaries; frequent splitting/merging is discouraged.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 1,\ 1,\ 1,\ 0\} \quad \Rightarrow \quad T = \{1,\ 1,\ 1,\ 1,\ 1,\ 0\}.$$

Team Topologies demonstrates one of the strongest alignments with centaur development among contemporary organizational models. It provides clear boundaries, strong local autonomy, modular interaction surfaces, and support for continuous delivery and automated governance. Its only major limitation is that it structurally assumes multi-person human teams; the topology does not natively support micro-unit (1–3 person) entities, which limits compatibility with centaur-style organizational decomposition.

SPOTIFY MODEL

The Spotify Model popularized the concepts of squads, tribes, chapters, and guilds as a way to scale Agile practices. Squads operate as semi-autonomous cross-functional teams; tribes group squads for alignment; chapters provide functional cohesion; and guilds offer voluntary knowledge-sharing communities. Although widely referenced, the Spotify Model is less a formal organizational framework and more an emergent pattern described through case studies. Nonetheless, it remains influential and represents a hybrid of autonomous product teams with overlaying communities of practice. Its reliance on multi-person squads and synchronous rituals limits compatibility with centaur micro-units, while its emphasis on autonomy and alignment structures produces moderate alignment on several dimensions.

ID	Val	Rationale
O1.1	1	Squads have substantial authority over local product and technical decisions.
O1.2	1	Squads operate within relatively flat structures at the tribe level.
O1.3	1	Squads may choose their own work practices (Scrum, Kanban, XP variants).
O2.1	1	Most day-to-day decisions are localized within squads.
O2.2	0	Chapter leads and tribe leadership impose some centralized influence on priorities and standards.
O2.3	1	The model supports decentralized decision-making under uncertainty through autonomous squads.
O3.1	0	Cross-squad alignment commonly occurs through ceremonies, chapter meetings, and synchronous coordination.
O3.2	0	Interaction patterns rely on interpersonal collaboration more than contract-driven modularity.
O3.3	0	Model does not prescribe explicit cross-team interface contracts or platform boundaries.
O4.1	0	Governance is largely manual and role-driven (tribe leads, chapter leads).
O4.2	1	Squads typically own quality signals and internal risk management.
O4.3	0	Automated or continuous governance is not structurally inherent.
O5.1	1	The model supports rapid delivery through squad autonomy.
O5.2	1	Prioritization can shift quickly within squads.
O5.3	1	Little batching is forced at the squad level; squads iterate continuously.
O6.1	0	The model assumes squads of 6–12 people; micro-units are not recognized.
O6.2	0	Team size expectations and roles prevent 1–3 person units.
O6.3	0	The structural model assumes stable squads; dynamic merging/splitting is not part of the design.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 2/3,\ 0,\ 1/3,\ 1,\ 0\}, \qquad T = \{1,\ 1,\ 0,\ 0.5,\ 1,\ 0\}.$$

The Spotify Model shows strong alignment with centaur needs in autonomy (O1), local decision-making (O2), and high-frequency iteration (O5). However, its cross-team coordination practices are largely synchronous and interpersonal (O3), governance is manual rather than automated (O4), and the model strongly assumes multi-person squads instead of micro-units (O6). As a result, it offers partial compatibility overall, with several dimensions that misalign with the coordination characteristics of centaur development.

PLATFORM ORGANIZATION

A platform organization structures work around a set of platform teams that provide reusable services, infrastructure, and capabilities to product or stream-aligned teams. Platform teams reduce cognitive load by enabling other teams to consume capabilities via self-service APIs, tools, or automated workflows. This structure increasingly appears in large-scale technology firms and aligns closely with modular architectures, automated governance, and cross-boundary efficiency–all relevant to centaur development. The primary limitation is that both platform and consuming teams are still assumed to be multi-person human teams rather than micro-units.

ID	Val	Rationale
O1.1	1	Product teams and platform teams have significant autonomy over local decisions.
O1.2	1	Hierarchy is relatively flat; platform teams support, not control, stream-aligned teams.
O1.3	1	Teams can choose workflows as long as they conform to platform interfaces.
O2.1	1	Decision rights for product direction and technical evolution are pushed into the stream-aligned teams.
O2.2	1	Platform teams operate as service providers; central decision-making is limited.
O2.3	1	The model explicitly supports decentralized decision-making under uncertainty.
O3.1	1	Platform interfaces (APIs, self-service tooling) serve as modular boundaries across units.
O3.2	1	Dependencies are handled through contract-based, asynchronous interactions.
O3.3	1	Platformization reduces synchronous cross-team coordination dramatically.
O4.1	1	Governance can be embedded into platform services (policy-as-code, automated guardrails).
O4.2	1	Risk and quality signals are often built directly into platform capabilities.
O4.3	1	Continuous, automated governance emerges naturally from platform interfaces.
O5.1	1	Product teams iterate independently; platforms minimize bottlenecks.
O5.2	1	Priorities can shift rapidly without requiring cross-team negotiation.
O5.3	1	Batching is reduced by self-service models.
O6.1	0	Teams are assumed to be multi-person; micro-units are not explicitly supported.
O6.2	0	Platform/stream team designs assume stable team structures rather than 1–3 person units.
O6.3	0	Dynamic merging/splitting at the micro-unit scale is not a design principle.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 1,\ 1,\ 1,\ 0\}, \qquad T = \{1,\ 1,\ 1,\ 1,\ 1,\ 0\}.$$

The platform organization demonstrates *very strong alignment* with centaur development across autonomy, decentralization, modularity, governance, and iteration speed. Its primary limitation is the assumption of multi-person human teams rather than centaur-scale micro-units. Nevertheless, few organizational models score as highly across the core coordination dimensions relevant to human–AI unit work.

SOCIOCRACY 3.0 (S3.0)

Sociocracy 3.0 (S3.0) is a collection of patterns for self-governance, decision making, and organizational design. It extends classical sociocracy with Lean and Agile ideas, emphasizing equivalence in decision-making, distributed authority, and dynamic role formation. S3.0 promotes autonomy and decentralized governance, making it partially compatible with centaur development; however, its reliance on consent-based group decision processes and circle structures assumes multi-person human groups rather than micro-units. Several coordination mechanisms remain inherently synchronous and human-centric, limiting fit with agentic workflows.

ID	Val	Rationale
O1.1	1	Circles (teams) have strong local authority to make decisions within their domain.
O1.2	1	Hierarchical layering is minimized; authority is distributed through linked circles.
O1.3	1	Circles can adapt their internal processes, roles, and domain responsibilities autonomously.
O2.1	1	Decision-making is intentionally delegated to the most local circle with sufficient context.
O2.2	1	There is minimal reliance on centralized PMOs or committees; governance is distributed.
O2.3	1	The consent principle supports decentralized, context-driven decision-making under uncertainty.
O3.1	0	Cross-circle coordination relies heavily on link roles and consent-based meetings, not modular boundaries.
O3.2	0	Dependencies require synchronous participation (double-linking, inter-circle meetings).
O3.3	0	S3.0 does not promote platformization or contract-driven alignment across groups.
O4.1	0	Governance processes are meeting-heavy and primarily human-centric.
O4.2	1	Quality and risk are often handled within circles through distributed roles and responsibilities.
O4.3	0	Continuous automated governance is not structurally part of S3.0.
O5.1	1	Circles can adapt their planning cadence as needed; planning is lightweight.
O5.2	1	Priorities can shift quickly through local consent processes.
O5.3	1	Decisions are not batched; changes occur as needed.
O6.1	0	S3.0 assumes circles of multiple humans participating in consent decision-making; micro-units are not modeled.
O6.2	0	Role distribution and linking mechanisms assume multi-person teams.
O6.3	0	Circle structures are stable; dynamic merging or splitting of 1–3 person units is not a core design principle.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 0,\ 1/3,\ 1,\ 0\}, \qquad T = \{1,\ 1,\ 0,\ 0.5,\ 1,\ 0\}.$$

Sociocracy 3.0 demonstrates strong alignment with centaur needs in autonomy (O1), decentralization (O2), and adaptability to high iteration cadence (O5). However, it is weakened by meeting-centric cross-boundary coordination (O3), manual governance structures (O4), and an explicit dependence on multi-person human circles (O6). As a result, S3.0 provides only a partial match to the coordination patterns required by human–AI micro-units.

HOLACRACY

Holacracy is a formalized self-management system introduced by Robertson. It distributes authority through explicitly defined roles, circles, and structured governance processes. While Holacracy aggressively decentralizes decision-making and provides high local autonomy, its coordination mechanisms rely on highly prescriptive, synchronous governance rituals (tactical meetings, governance meetings, integrative decision-making). These human-centric processes impose significant communication overhead and are structurally incompatible with the asynchronous, high-frequency iteration patterns of centaur micro-units.

ID	Val	Rationale
O1.1	1	Roles and circles have substantial autonomy to act within their domains.
O1.2	1	Hierarchy is intentionally minimized; authority is distributed to roles rather than managers.
O1.3	1	Circles can adapt local processes and governance rules within the Holacracy constitution.
O2.1	1	Decision-making authority is decentralized to circle-level roles.
O2.2	1	Centralized oversight is minimal; governance flows outward, not upward.
O2.3	1	Holacracy's structure is explicitly designed for decentralized action under uncertainty.
O3.1	0	Cross-circle alignment requires structured governance meetings with mandatory synchronous participation.
O3.2	0	Dependencies are handled through integrative decision-making rather than contract-based boundaries.
O3.3	0	Holacracy does not emphasize modular interfaces between circles; alignment is interpersonal.
O4.1	0	Governance is entirely meeting-based and manual.
O4.2	1	Quality and risk management can be embedded locally within roles.
O4.3	0	Continuous automated governance is absent from the structural model.
O5.1	1	Roles can adapt and evolve quickly through governance processes.
O5.2	1	Priorities can shift dynamically because roles change based on current tensions.
O5.3	1	Decisions are processed iteratively in governance cycles rather than batched.
O6.1	0	Holacracy assumes circles consisting of multiple human roles; micro-units are not modeled.
O6.2	0	Team size flexibility is low due to the rule-bound nature of governance meetings.
O6.3	0	Circles are not designed for rapid splitting/merging of 1–3 person entities.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 0,\ 1/3,\ 1,\ 0\}, \qquad T = \{1,\ 1,\ 0,\ 0.5,\ 1,\ 0\}.$$

Holacracy provides strong alignment with centaur development in autonomy (O1), decentralization (O2), and fast iteration (O5). However, the model is highly meeting-centric and depends on synchronous, human-only interaction patterns for cross-boundary alignment and governance (O3, O4). It also lacks support for micro-unit structures (O6). These factors make Holacracy partially compatible but structurally limited under a centaur paradigm.

TEAL ORGANIZATIONS

Teal organizations, as described by Laloux, emphasize three core principles: self-management, evolutionary purpose, and wholeness. They reject traditional hierarchy entirely in favor of decentralized authority and fluid role formation. Teams are self-organizing, individuals hold multiple dynamic roles, and decision making follows an "advice process" rather than formal governance bodies. This creates extremely high autonomy and decentralization—features that align well with centaur development. However, Teal organizations often rely heavily on interpersonal, synchronous consultation and do not provide strong structural support for modular interaction surfaces, automated governance, or micro-unit organization. As a result, alignment is mixed.

ID	Val	Rationale
O1.1	1	Authority is radically decentralized; individuals and teams self-manage without hierarchical approval.
O1.2	1	There are no managerial layers; communication channels are inherently flat.
O1.3	1	Units can freely adjust local processes and self-organize structures.
O2.1	1	Decisions are made where information resides via the "advice process."
O2.2	1	There are no PMOs or centralized decision committees; governance is distributed.
O2.3	1	Self-management explicitly supports decentralized decision-making under uncertainty.
O3.1	0	Cross-team alignment relies heavily on interpersonal advice-seeking, not modular boundaries or APIs.
O3.2	0	Dependencies require consultations, which are synchronous and human-dependent.
O3.3	0	Teal organizations do not prescribe contract-based or platform-style interfaces.
O4.1	0	Governance is human-driven (advice, consent) and not automated.
O4.2	1	Teams handle risk and quality locally through self-management principles.
O4.3	0	Continuous automated governance is not part of Teal organizational design.
O5.1	1	Planning is fluid and continuous; units respond quickly to new information.
O5.2	1	Priorities shift rapidly because authority is localized and contextual.
O5.3	1	No batching is imposed; decisions can be made immediately via advice process.
O6.1	0	Micro-units are not a structural concept; work assumes multiple humans in fluid roles.
O6.2	0	There is no support for 1–3 person formal units; Teal roles are fluid but not micro-team based.
O6.3	0	Structural stability is assumed; splitting/merging micro-units is not modeled.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 0,\ 1/3,\ 1,\ 0\}, \qquad T = \{1,\ 1,\ 0,\ 0.5,\ 1,\ 0\}.$$

Teal organizations align very well with centaur development in autonomy (O1), decentralization (O2), and rapid iteration (O5). However, they misalign with the requirements for minimal cross-boundary coordination (O3), automated governance (O4), and the centaur-compatible concept of micro-units (O6). In practice, Teal structures depend heavily on human-centric synchronization mechanisms and lack the modularity and governance automation needed for large-scale centaur ecosystems.

RENDANHEYI (HAIER MICRO-ENTERPRISE MODEL)

Rendanheyi, developed within the Haier Group, is one of the most thoroughly documented micro-enterprise organizational models in practice. The structure breaks the organization into hundreds or thousands of autonomous micro-units (often 1–10 people), each with its own P&L accountability, decision rights, and direct customer or internal-market relationships. Coordination occurs through contract-like interfaces (platform and service marketplaces) rather than hierarchical control. Governance and alignment mechanisms are embedded in platforms, performance contracts, and transparent market interactions. This makes Rendanheyi uniquely aligned with centaur development, which assumes autonomous micro-units, decentralized decision rights, modular boundaries, and automated governance through platform interface layers.

ID	Val	Rationale
O1.1	1	Micro-enterprises possess full local authority for routine business and technical decisions.
O1.2	1	Hierarchical layers are minimized; platform units support autonomy rather than control it.
O1.3	1	Micro-enterprises choose their own workflows, processes, and business models.
O2.1	1	Decision-making occurs at the micro-enterprise level based on direct customer feedback.
O2.2	1	There is no central PMO or hierarchical committee controlling tactical decisions.
O2.3	1	The structure explicitly supports decentralized decision-making under uncertainty via internal markets.
O3.1	1	Coordination across units happens through platform contracts and marketplace mechanisms, not meetings.
O3.2	1	Dependencies are managed asynchronously via service interfaces (internal market transactions).
O3.3	1	Modularity is inherent: micro-enterprises interact through formalized service relationships.
O4.1	1	Governance is embedded in platform systems (contracts, performance tracking, transparent metrics).
O4.2	1	Risk and quality responsibilities are decentralized to each micro-enterprise.
O4.3	1	Continuous governance is enabled by platform-mediated monitoring and contracts.
O5.1	1	Micro-enterprises operate with continuous, real-time responsiveness to customer or internal-market signals.
O5.2	1	Priorities shift rapidly based on direct demand signals; no batch planning required.
O5.3	1	No structural batching; micro-enterprises iterate continuously.
O6.1	1	Micro-enterprises can be as small as 1–3 people, matching centaur micro-unit structure.
O6.2	1	Team sizes are flexible; micro-units form and dissolve based on market needs.
O6.3	1	Dynamic splitting/merging is part of the design; units evolve based on performance and market interactions.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 1,\ 1,\ 1,\ 1\}, \qquad T = \{1,\ 1,\ 1,\ 1,\ 1,\ 1\}.$$

Rendanheyi demonstrates the highest possible alignment with centaur development. It is the only widely deployed organizational structure that explicitly supports autonomous micro-units, decentralized authority, modular service interactions, automated governance through platform interfaces, continuous iteration, and dynamic recomposition of teams. As such, it provides a real-world analogue to how centaur-based organizations may evolve at scale.

NETWORKED / CELLULAR ORGANIZATIONS

Networked or cellular organizations structure work into small, semi-autonomous cells that operate with high internal cohesion and minimal hierarchy. Each cell is responsible for a coherent value stream or capability, and cells interact with each other through lightweight coordination mechanisms. While models vary (e.g., cellular manufacturing, networked firms, fractal organizations), the general principle is to organize the company as a distributed network of autonomous units connected through shared purpose, relationships, or contracts. This structure aligns well with centaur development because it naturally supports micro-unit autonomy, decentralized decision-making, and adaptive reconfiguration. However, implementations vary significantly, and some versions still rely on human-centric alignment rituals or lack strong platform-mediated governance.

ID	Val	Rationale
O1.1	1	Cells have strong local authority over domain-specific decisions.
O1.2	1	Hierarchical layering is minimal; cells communicate directly as peers.
O1.3	1	Cells choose their own workflows, tooling, and operational models.
O2.1	1	Decision rights are embedded in each cell; decisions are made close to the work.
O2.2	1	There is typically no central PMO; coordination is distributed across cells.
O2.3	1	Cells adapt autonomously to uncertainty, consistent with decentralized decision-making theory.
O3.1	1	Cells often coordinate through contract-like agreements or loosely coupled interaction standards.
O3.2	1	Dependencies can be handled asynchronously through defined interfaces or negotiated relationships.
O3.3	1	Networked structures emphasize modularity and weak coupling between units.
O4.1	0	Automated governance is not guaranteed; implementations vary widely.
O4.2	1	Cells typically embed quality and risk controls locally.
O4.3	0	Many networked organizations rely on human-driven trust and relationships rather than automated policies.
O5.1	1	Cells adapt continually based on local signals or shared system needs.
O5.2	1	Priorities shift rapidly within each cell without requiring cross-unit planning cycles.
O5.3	1	Minimal batching: decisions occur immediately within each unit.
O6.1	1	Cells can be very small (2–5 people), and micro-unit variations exist in several implementations.
O6.2	1	Team size and structure are flexible; many implementations allow 1–3 person cells.
O6.3	1	Cells may split or recombine dynamically as workloads or capabilities evolve.

INDICATOR TABLE WITH RATIONALES.

SCORES.

$$R = \{1,\ 1,\ 1,\ 2/3,\ 1,\ 1\}, \qquad T = \{1,\ 1,\ 1,\ 0.5,\ 1,\ 1\}.$$

Networked or cellular organizations exhibit very strong structural compatibility with centaur development. They support local autonomy, decentralized decision- making, modular interactions, high iteration speed, and flexible recomposition of units. The only partial limitation is governance (O4), which varies significantly across implementations and is often human-driven rather than platform-automated. Despite this, the structure remains one of the closest matches to the coordination and autonomy requirements of human–AI micro-unit work.

SUMMARY OF ORGANIZATIONAL MODEL COMPATIBILITY SCORES

Table 4 consolidates the ternary compatibility scores for all twelve organizational models across the six centaur-relevant dimensions (O1–O6). Each score is derived mechanically from the indicator values presented in the preceding subsections, ensuring that the results are transparent and reproducible. The table highlights substantial variation in how different organizational structures distribute authority, manage cross-boundary coordination, integrate governance, adapt to high-frequency iteration, and accommodate micro-unit topologies.

High scores (1) indicate strong compatibility with centaur development–namely organizational conditions that support autonomy, local decision-making, asynchronous coordination, embedded governance, rapid iteration, and flexible team topology. Mid-range scores (0.5) represent partial or context-dependent alignment. Low scores (0) denote structural misalignment rooted in hierarchical control, human-centric interaction mechanisms, synchronous coordination, or fixed assumptions about team size.

Table 4: *Organizational Model Compatibility Scores (O1–O6)*

Organizational Model	O1	O2	O3	O4	O5	O6
Functional Hierarchy	0	0	0	0	0	0
Divisional (M-Form)	0	0	0	0	0	0
Matrix Organization	0	0	0	0	0	0
Cross-Functional Product Teams	1	1	0	0.5	1	0
Team Topologies	1	1	1	1	1	0
Spotify Model	1	1	0	0.5	1	0
Platform Organization	1	1	1	1	1	0
Sociocracy 3.0	1	1	0	0.5	1	0
Holacracy	1	1	0	0.5	1	0
Teal Organizations	1	1	0	0.5	1	0
Rendanheyi (Micro-Enterprise)	1	1	1	1	1	1
Networked / Cellular Organizations	1	1	1	0.5	1	1

ANALYSIS. The results reveal a clear structural divide between traditional hierarchical organizations and more modern or modular forms. Functional, divisional, and matrix structures exhibit complete misalignment across all dimensions, reflecting their dependence on centralized authority, synchronous communication, layered approval processes, and large multi-role teams. Contemporary cross-functional product teams offer substantial improvements in autonomy (O1), decentralization (O2), and iteration adaptability (O5), but remain constrained by interpersonal alignment mechanisms and an implicit assumption that teams must be multi-person (O6).

Team Topologies, Platform Organizations, and Networked/Cellular structures demonstrate the strongest overall compatibility with centaur development. These models combine decentralized authority, modular boundaries, reduced cognitive load, and embedded or automatable governance mechanisms–structural properties that support high-frequency iteration and minimize cross-boundary coordination costs. Rendanheyi stands out as the only model that fully satisfies all six dimensions, offering a real-world example of an organizational design that explicitly supports autonomous micro-units, platform-mediated governance, internal markets, and dynamic recomposition of teams.

In contrast, Sociocracy 3.0, Holacracy, and Teal organizations excel in autonomy and decentralization but depend heavily on synchronous, human-centric governance rituals and lack explicit support for modular boundaries or automated alignment mechanisms. These characteristics limit their suitability for organizations adopting centaur-based human–AI workflows. Overall, the analysis suggests that organizational structures optimized for modularity, autonomy, and continuous governance provide the best fit for centaur development, while traditional hierarchical models remain structurally incompatible.

CASE ILLUSTRATION

To complement the theoretical analysis presented so far, this section examines a single illustrative case in which a non-trivial software system was developed by a senior engineer working in partnership with an AI coding agent (Blomgren, 2025a). The system, `lockd`, was co-developed by one of the co-authors of this essay together with an AI agent during approximately five weeks of limited spare time. The purpose of this case is not to generalize from anecdote, but to demonstrate concretely how the coordination dynamics of centaur development manifest when a single human–AI unit undertakes work that would traditionally require an entire team.

LOCKD IS A COORDINATION SERVICE — a type of backend system that provides foundational building blocks for many modern distributed applications. Such systems typically support capabilities that let other programs coordinate shared work safely, such as: exclusive locks, shared state, structured storage, simple queries, and lightweight messaging. These capabilities form the "plumbing" of distributed computing and are normally provided by multiple separate systems (e.g., object stores, document stores, lock managers, and queueing systems).

In contrast, `lockd` combines several of these foundational capabilities into a single, cohesive system. It supports:

- **Exclusive leases** for safe coordination between distributed processes;
- **Structured state storage** that lets clients store and update arbitrary JSON-like data;
- **Simple querying** over stored documents;
- **A lightweight at-least-once queue**, enabling task dispatch and background processing;
- **Multiple storage backends**, meaning the system can run on a laptop, cloud object stores, or local filesystems without modification.

To an experienced distributed-systems engineer, the integration of these elements into a single, coherent system is immediately recognizable as complex work: it spans concurrency control, storage abstraction, error handling, consistency considerations, and multi-backend operational concerns. To non-specialists, it is helpful to view `lockd` as an "infrastructure kernel" that applications can trust for safe coordination—the sort of component that, in industry, would typically be developed and maintained by a small team of senior engineers.

WHY *LOCKD* IS NON-TRIVIAL

From an engineering perspective, `lockd` reflects several forms of complexity that normally require multiple specialists:

- **Distributed systems complexity:** building safe coordination primitives requires careful handling of race conditions, contention, and failure scenarios;
- **Storage and backend abstraction:** supporting multiple backend engines requires designing a clean, unified abstraction layer so that all features behave consistently across implementations;
- **Query processing:** even simple querying requires parsers, evaluators, and correct traversal of structured state;
- **Queue semantics:** implementing at-least-once delivery, visibility timeouts, and retry logic is conceptually simple but operationally subtle;
- **Integration quality:** testing across various backends, ensuring consistent behaviour, and producing one coherent binary all demand continued architectural discipline.

None of these elements are individually exotic. What is non-trivial is that they were *all* integrated into a single system with coherent design, consistent semantics, and comprehensive test coverage—the kind of integration work that requires breadth of experience and careful coordination if done by a human team.

HUMAN–AI COLLABORATION

According to the repository's own documentation, nearly all code in `lockd` was generated by an AI coding agent, while the human senior engineer (the co-author of this essay) acted as the system's architect, governor, and reviewer. The human contributor:

- defined the architecture and its evolution;
- framed requirements and constraints;
- identified design flaws and corrected incorrect AI output;
- ensured consistency across components;
- created or refined the verification mechanisms—tests, integration suites, and other guardrails.

The AI coding agent, in turn:

- generated and iteratively refined large portions of code;
- produced boilerplate and glue logic quickly;
- assisted with test scaffolding and repeated structural transformations;
- explored alternative implementations to satisfy constraints and performance goals.

This division of labour mirrors the centaur pattern observed in other domains: the human provides expertise, judgment, context, and responsibility; the AI provides speed, exhaustiveness, and the ability to transform entire components in seconds.

WHY THIS IS AN ILLUSTRATIVE EXAMPLE

The fact that a senior engineer—working only in spare time over roughly five weeks—could produce a system of this breadth and integration is itself indicative of a dramatic shift in effective capability. In traditional settings, a project such as `lockd` would typically involve:

- one or two engineers specializing in distributed coordination;
- another specializing in storage abstractions;
- perhaps a separate engineer focusing on queueing semantics or API design;
- someone to maintain integration and test suites;
- and a team lead or architect to maintain coherence across efforts.

The centaur unit (human + AI system) replaced this entire multi-role pipeline. This does not imply that the AI replaces the team; rather, it shows that a human expert, when augmented by a capable AI agent, can handle far more surface area than was possible before. The AI accelerates execution and exploration, while the human ensures correctness, feasibility, and conceptual integrity.

IMPLICATIONS FOR THE ANALYSIS

The `lockd` case does not provide controlled experimental evidence, nor does it demonstrate how multiple centaur units coordinate at scale. However, it does serve as a concrete—and publicly inspectable—example of a centaur unit performing work historically distributed across a team. This aligns with the central claim of this essay: that the unit of effective action in software development is shifting. Coordination structures built around multi-person teams will need to adapt as AI agents allow senior engineers to operate as high-capability micro-units, provided that sufficient governance and oversight are maintained.

The case therefore functions as an existence proof: under realistic constraints, a senior engineer augmented by an AI coding agent can deliver a cohesive, complex, multi-component system in a timeframe that would have been implausible even for a highly skilled individual only a few years ago.

WHY DISTRIBUTED COORDINATION SYSTEMS ARE HARD

Readers without a background in distributed systems may reasonably ask why a system such as `lockd` is considered non-trivial. Coordination services sit at the foundation of modern computing: they mediate shared state, ensure correctness under concurrency, and allow independent processes to collaborate reliably. Although the individual components of such systems can be explained simply, building them correctly is challenging because they combine several categories of difficulty that compound in practice.

CONCURRENCY IS INHERENTLY COMPLEX. Whenever multiple programs act on shared data, subtle timing differences can lead to inconsistent outcomes. Coordination systems must prevent race conditions, lost updates, and conflicting operations, even when many clients act simultaneously. This requires careful reasoning about atomicity, ordering, and isolation–properties that cannot be retrofitted easily once the system exists.

FAILURE IS THE NORMAL CASE. Distributed systems cannot assume a stable environment. Storage devices fail, network connections drop, processes restart, and clocks drift. Coordination logic must remain correct across all such failures, which means building recovery paths, retry logic, and idempotent operations. Designing these cases is often more work than implementing the "happy path" itself.

THE SYSTEM MUST BE PREDICTABLE ACROSS BACKENDS. A coordination service that supports multiple storage engines must present one coherent behavioral model across them, even if the underlying infrastructure behaves differently. This introduces tensions between abstraction and performance: the interface must hide backend differences without forcing every backend into a lowest-common-denominator model.

CORRECTNESS MUST BE DEMONSTRATED, NOT ASSUMED. Testing distributed coordination is fundamentally more difficult than testing a single-threaded component. Integration tests must simulate failures, restarts, interleavings, and edge-case interactions across subsystems. This is why such systems often include substantial test suites, harnesses, and watchdog mechanisms.

SMALL DESIGN CHOICES HAVE SYSTEM-WIDE CONSEQUENCES. Whether to retry an operation, how to detect conflicts, how to track metadata, or when to expire leases–these are small local decisions that can affect global consistency and performance. Coordination systems require a coherent architecture because inconsistencies in one part can undermine correctness elsewhere.

REAL-WORLD REQUIREMENTS MULTIPLY COMPLEXITY. Encryption, authentication, observability, performance constraints, container semantics, and cloud compatibility all add layers of engineering detail that must interlock cleanly. Mature coordination services are not just algorithms but operationally robust systems.

Taken together, these factors explain why coordination systems are typically developed by experienced teams over long time horizons. They require sustained architectural reasoning, precise engineering, rigorous testing, and careful handling of edge conditions. That a centaur unit–one senior engineer working with an AI agent–could construct such a system in a few weeks highlights both the accelerating potential of human–AI collaboration and the need to reassess how software development work is organized in an era where execution is no longer the primary bottleneck.

IMPLICATIONS AND FUTURE DIRECTIONS

The preceding sections examined how delivery frameworks and organizational models align with the coordination demands of centaur development and illustrated these dynamics through a concrete case. We now synthesize these findings and outline implications for software delivery, organizational design, and future research. While this analysis does not claim universal prescriptions, it reveals a set of structural pressures that organizations and practitioners will increasingly confront as human–AI collaboration matures.

REVISITING THE CENTRAL QUESTION

This essay began with a simple but consequential question: *What happens to software delivery frameworks and organizational structures when a single senior engineer, augmented by agentic AI tools, can accomplish work that previously required a multi-person team?* The evaluations in Evaluation of Delivery Frameworks (p. 14) and Evaluation of Organizational Models (p. 25) indicate that this shift fundamentally alters the locus of coordination. In traditional settings, coordination overhead arises primarily from interactions *within* teams. In centaur development, this overhead is reduced or eliminated inside a human–AI unit, while coordination *between* units, systems, and organizational boundaries becomes the central challenge.

THE EMERGENCE OF THE CENTAUR UNIT

Across the analysis, the *centaur unit*—a human expert working closely with one or more AI agents—emerges as a meaningful abstraction. A centaur unit collapses what were previously distinct roles (architect, developer, reviewer, tester, operations engineer) into a single cognitive locus supported by machine-driven execution and exploration. The effectiveness of this unit depends on human judgment, verification, and direction, but its execution capacity is amplified by AI agents that can produce and refine artifacts rapidly and at scale.

The viability of centaur units does not imply that human teams disappear, nor that AI replaces expertise. Instead, the centaur unit becomes a new "unit of production" in software development, existing alongside human-only teams and automated systems. This reframes coordination problems: instead of managing complex dynamics within teams, organizations must design structures that allow many autonomous, high-capability units to interact safely and coherently.

IMPLICATIONS FOR SOFTWARE DELIVERY FRAMEWORKS

The delivery framework evaluations in Evaluation of Delivery Frameworks (p. 14) show a marked pattern. Frameworks designed around synchronous rituals, fixed iteration cycles, and rigid role structures (Scrum, SAFe, Waterfall) exhibit low compatibility with centaur development. Their coordination mechanisms presuppose stable, medium-sized human teams and do not translate cleanly to settings where execution is performed largely by automated systems under the guidance of a single senior engineer.

In contrast, frameworks emphasizing continuous flow, lightweight practices, automation, and rapid experimentation (Kanban, Lean Startup, DevOps/SRE) demonstrate strong compatibility. These approaches treat delivery not as a sequence of human meetings but as a continuous socio-technical feedback system, which aligns well with human–AI workflows that iterate at high frequency. Automated governance and verification (D4) emerge as central pillars: when AI agents produce a large volume of artifacts, test suites, pipelines, and observability systems become the primary means of ensuring correctness and safety.

IMPLICATIONS FOR ORGANIZATIONAL DESIGN

Evaluation of Organizational Models (p. 25) reveals a similarly strong pattern. Traditional hierarchical and matrix structures score poorly across all centaur-relevant dimensions. Their dependence on centralized authority, synchronous cross-functional alignment, and fixed team sizes makes them structurally misaligned with autonomous micro-units.

Modern organizational forms that emphasize modularity, platformization, and decentralized decision-making (Team Topologies, Platform Organizations, Networked/Cellular Organizations) rank significantly higher. These models create clear interaction surfaces between units, reduce

coordination overhead, and support continuous iteration. The Rendanheyi micro-enterprise model stands out as a real-world example of an organization built around autonomous, P&L-bearing micro-units–an architecture conceptually similar to how centaur units might operate at scale.

A key implication is that organizational design in the centaur era shifts from *managing teams* to *managing boundaries*. Platforms, internal markets, service contracts, and architectural modularity become organizational tools as much as technical ones.

THE NEW COORDINATION PROBLEM

As execution becomes cheap and fast within a centaur unit, the primary challenges move to the inter-unit level:

- **How do autonomous units coordinate without recreating human team overhead?**
- **What structures ensure alignment without synchronous meetings?**
- **How do platform teams provide boundary stability and governance at scale?**
- **How do organizations prevent fragmentation as autonomy increases?**

These are not new challenges in organizational theory, but centaur development increases their importance. As the internal cohesion of each centaur unit grows, so does the need for clear external boundaries and automated governance mechanisms.

HUMAN WORK IN THE CENTAUR ERA

Centaur development does not diminish the relevance of human expertise. If anything, it elevates the value of senior engineers who can provide architectural judgment, contextual reasoning, risk assessment, and verification. The human role shifts from performing execution to steering it, from producing artifacts to governing their correctness, and from coordinating with teammates to coordinating across systems.

This raises practical questions for capability development. How do junior engineers grow in a world where much execution work is delegated to AI? Organizations will need to invest in apprenticeship pathways, pairing structures, and opportunities for juniors to practice judgment under guidance.

OPEN QUESTIONS AND FUTURE RESEARCH

While the analysis in this paper highlights several structural patterns, it also surfaces open questions requiring further empirical and theoretical work. Among them:

- **Multi-centaur coordination:** How do many centaur units collaborate on shared architectures without recreating team overhead?
- **Governance and safety:** What mechanisms ensure that AI-driven execution remains safe, verifiable, and aligned with organizational goals?
- **Boundary design:** Which architectural patterns (APIs, service contracts, platforms) best support high-autonomy units?
- **Organizational resilience:** How do centaur-heavy organizations handle failure, turnover, and knowledge transfer?
- **Skill development and human capital:** How will the distribution of expertise evolve when execution is augmented by AI?
- **Coordination economics:** How will organizations trade off the benefits of autonomy against the risks of fragmentation?

These questions underscore that centaur development represents not only a shift in execution but a broader transformation in how software work is organized, aligned, and governed.

SUMMARY. Centaur development alters the economics of software engineering: execution becomes cheap, exploration accelerates, and coordination within units largely disappears. As a result, coordination between units, automated governance, modular architecture, and organizational boundary design become the central challenges. The frameworks and organizational models most compatible with this future are those that optimize for autonomy, modularity, and continuous, automation-friendly workflows. Much remains to be explored, but the trends identified here suggest a significant reconfiguration of software delivery and organizational design in the years ahead.

CONCLUSION

Across the history of software engineering, the constraints shaping delivery frameworks and organizational structures have been remarkably stable. They have all assumed that software is built by groups of humans whose primary limitation is cognitive and communicative capacity. Methods and structures—from functional hierarchies to Scrum, from project governance to scaled frameworks—arose as means of coordinating multiple people working together under these constraints. The central contribution of this essay is to show that this assumption, while historically reasonable, is no longer adequate to describe the actual possibilities of software work in the post–2025 era of agentic AI systems.

The analysis presented here demonstrates that when a single senior engineer, augmented by capable AI agents, can meaningfully perform work previously requiring an entire team, the coordination landscape shifts. Many of the mechanisms that justified rituals, roles, committees, meetings, and procedural governance no longer operate at their historical efficiency frontier. Within a centaur unit, execution is fast, iteration is cheap, and intra-unit coordination costs are near zero. Under such conditions, many assumptions embedded in traditional delivery frameworks and organizational models become structural misfits rather than enablers.

The delivery-framework evaluation revealed that methods built around synchronous team ceremonies, fixed iteration cycles, and rigid role boundaries (e.g., Scrum, SAFe, PRINCE2, Waterfall) offer limited compatibility with centaur development. Their coordination primitives assume multiple humans negotiating a shared plan in real time, an assumption that no longer holds when the unit of production becomes a human–AI hybrid. In contrast, lightweight, flow-oriented, automation-first frameworks (Kanban, Lean Startup, DevOps/SRE) map more closely to centaur development because they replace ceremony with feedback loops and role specialization with capability-based autonomy. They treat the delivery system as a continuously evolving socio-technical feedback network rather than a schedule of events.

The organizational-model evaluation reached parallel conclusions. Traditional hierarchical, divisional, and matrix structures rely on multi-layered authorities, synchronous cross-team alignment, and stable multi-human teams. These structures scored uniformly poorly across autonomy, decentralization, cross-boundary efficiency, governance integration, and micro-unit accommodation. In contrast, modern modular forms—Team Topologies, Platform Organizations, Networked/Cellular structures—and especially Rendanheyi's micro-enterprise model showed strong compatibility with centaur development. These models are built not around managing teams but around managing boundaries: APIs, service contracts, marketplaces, and platform interfaces. As execution accelerates, boundaries—not ceremonies—become the primary coordination mechanism.

The case illustration of *lockd* served to ground the theoretical analysis in a concrete, publicly available artifact. It showed how a single senior engineer, working in partnership with an AI agent, can produce in weeks a system that spans distributed coordination, storage abstraction, queueing, querying, encryption, and integration across multiple backends. This does not prove that all teams or all domains can work this way; nor does it suggest that AI replaces teams. Instead, it demonstrates that the assumptions underpinning traditional coordination models no longer hold universally. Where execution can be delegated to AI, the human becomes the architect, verifier, and governor, and the human–AI unit becomes a micro-enterprise capable of end-to-end delivery.

This shift is not merely a technical improvement; it is a structural one. It reconfigures the economics of software development. Historically, the costliest activities were those involving human execution and human coordination. Now, for certain classes of work, execution is no longer the bottleneck. Coordination, verification, and boundary management become the limiting factors. Software engineering moves from being a predominantly execution-constrained discipline to being a coordination-constrained one. This echoes long-standing insights from organizational theory, but in a new technological context where the unit of effective production is no longer a team but the centaur unit.

This transformation also introduces new challenges. Multi-centaur coordination, boundary design, automated governance, and organizational resilience under high-autonomy conditions are complex questions that require further research. Moreover, the cultivation of human expertise—especially for junior engineers—becomes more

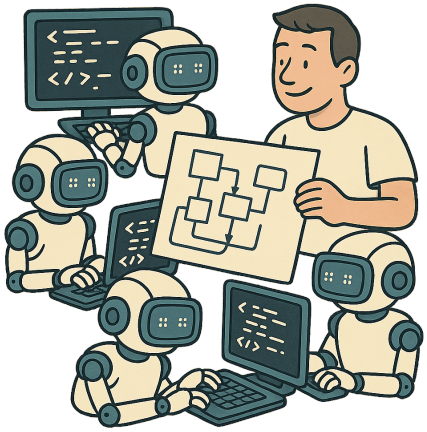
important, not less. AI expands what experts can do, but it does not teach judgment, architectural reasoning, or responsibility. These remain irreducibly human competencies and must be developed intentionally.

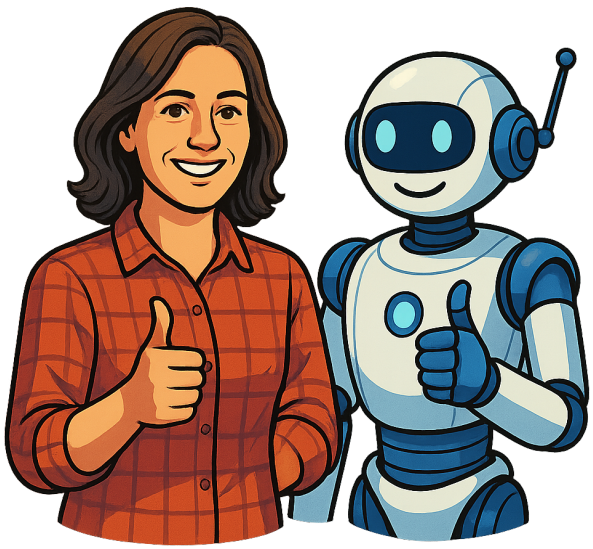
Yet the potential is profound. The centaur unit may represent one of the most significant shifts in the history of software engineering–comparable in impact to the emergence of high-level languages, interactive computing, or the rise of the internet. It does not diminish human creativity or agency; it amplifies it. It offers a future in which small, highly capable human–AI units can build, explore, and iterate at speeds previously unattainable, while platforms, architectures, and organizational structures evolve to support them. Realizing this potential responsibly will require new forms of governance, new delivery practices, and new organizational designs, but the foundations outlined in this essay provide a starting point for that exploration.

As software systems grow in ambition and complexity, and as AI systems grow in capability, the question is no longer whether centaur development will reshape how we build and organize software work, but how we choose to shape the structures that support it. This essay offers one framework for understanding that choice and invites researchers, practitioners, and leaders to continue the conversation.

CAVEATS: HERE BE DRAGONS

The results described in Case Illustration (p. 39) regarding `lockd` (Blomgren, 2025a)—and other projects such as pkt.systems/pslog and pkt.systems/kryptograf—are not a free lunch. They rely on a way of working that is still unevenly distributed: the ability to externalise intent clearly, to design verification harnesses that keep fallible agents on a safe path, and to treat AI output as a raw material for disciplined engineering rather than as an oracle. In other words, centaur development is as much a *skill* and a *mental model* as it is a tooling choice. Two engineers with access to the same state-of-the-art agent platform will not produce the same outcomes if one has rebuilt their habits around iterative, test-first, agent-aware workflows and the other has not.





ABOUT THE AUTHOR(S)

This work was co-authored by Michel Blomgren and an AI capable of retrieval augmented generation (an LLM with tools). Michel Blomgren is a software engineer and solutions architect specializing in distributed systems, large-scale infrastructure, and operational automation. With nearly three decades of experience, he has designed and implemented fault-tolerant microservice architectures, distributed workflow systems, and cloud-native integration pipelines across both commercial and defense contexts.

His work includes developing Kubernetes controllers and operators, architecting resilient networking and platform services, and guiding organizations in adopting container orchestration, Infrastructure as Code, and secure software-delivery practices. Colleagues consistently note his ability to clarify complex system behavior and support effective cross-disciplinary collaboration.

Blomgren is also the author of the Outcome-Based Agile Framework (Blomgren, 2025b), reflecting his broader interest in how feedback loops, constraints, and adaptive planning can improve decision-making in complex sociotechnical systems. He is currently working as a senior consultant at Nion situated in Gothenburg, Sweden.

POSTSCRIPT: AFTER ACTION REVIEW

The After Action Review (AAR) is a structured reflective practice originating in the United States Army in the late 1970s. It was designed to help small units rapidly capture and disseminate operational learning in dynamic, high-stakes environments. Over time, it spread across NATO militaries and armed forces globally, becoming a standard mechanism for post-mission reflection in infantry, aviation, and logistics units. Civilian emergency services—including fire brigades, wildfire response teams, and crisis-management organizations—soon adopted it as well, valuing its emphasis on honest, non-punitive learning and rapid adaptation. Across these domains, the AAR serves as a disciplined method for transforming lived experience into actionable knowledge. In the context of centaur-style software development—where a single human–AI unit performs the work traditionally distributed across an entire team—the small-unit AAR offers the most fitting analogue: it focuses on cognition, judgment, and adaptation rather than formal process audits.

WHAT WAS SUPPOSED TO HAPPEN

The original aim of the `lockd` exercise was modest. The intention was to explore whether a complex coordination system—one that would ordinarily require a small team of senior engineers—could be meaningfully advanced by a single developer working in limited spare time, aided by an AI coding agent. The expectations were grounded in decades of conventional software delivery experience: perhaps a basic scaffold would emerge, perhaps one or two core capabilities would take shape, and perhaps the exercise would illuminate the practical limits of centaur-style development in non-trivial distributed systems. The anticipated outcome was exploratory rather than complete, more architectural sketch than functioning platform. The underlying assumption—consistent with classic coordination theory (Brooks, 1995; Conway, 1968) and contemporary accounts of human–AI teaming (Seeber, 2020; Schmutz, 2024)—was that architectural breadth, concurrency control, and integration complexity would impose a natural ceiling on solo development velocity.

WHAT ACTUALLY HAPPENED

The reality diverged sharply from these expectations. The first end-to-end iteration of `lockd` emerged not over months, but within roughly two weeks. All planned features materialized quickly, and then unplanned capabilities surfaced as well: a lightweight queue, an indexed search mechanism, a query language, and eventually preliminary support for XA transactions and federated coordination across distributed `lockd` "islands." From the perspective of the initial architecture, these capabilities seemed improbable for a single engineer to deliver so rapidly. Yet high-frequency iteration with the AI agent expanded the system's functional envelope far beyond the original scope. Features normally associated with multi-engineer teams became accessible to a single human–AI unit. As discussed in the main body of this essay, the limiting factor shifted from execution to the human's ability to maintain conceptual coherence across a rapidly evolving codebase.

WHY IT HAPPENED

This acceleration occurred because the centaur model collapses the execution bottleneck while preserving—and in places intensifying—the cognitive bottleneck. The AI coding agent removed much of the friction traditionally associated with implementation, refactoring, and exploratory branching. Structural changes could be attempted and reversed in minutes, and alternative designs sketched and evaluated in hours rather than weeks. This dynamic aligns with empirical findings on AI-assisted development (Ercin, 2025; Schreiber, 2025) and long-standing theories of distributed cognition (Hutchins, 1995). However, reduced execution cost produced a new form of cognitive load for the human half of the centaur. Instead of reasoning locally about a subsystem, the engineer had to sustain global coherence across an actively shifting architecture, acting simultaneously as architect, reviewer, integrator, and governor–roles that human–AI teaming research identifies as cognitively demanding under rapid system change (Seeber, 2020; Schmutz, 2024). The load was not heavier in quantity but broader in scope: an ambient, whole-system cognitive strain few existing practices prepare individuals to manage.

WHAT WE LEARNED

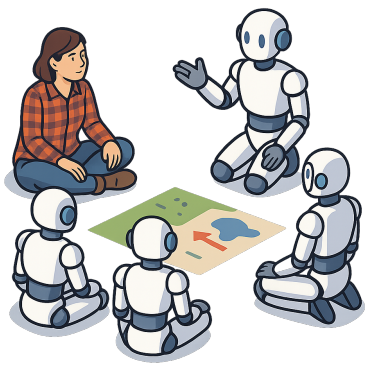
The most important lesson is not simply that more features can now be produced more quickly–striking though that fact may be. Rather, the underlying *configuration* of human work has shifted in a way that reinforces the core argument of this essay. In the technical analysis, we propose that agentic AI systems move software engineering from an execution-constrained discipline to a coordination-constrained one. Within a centaur unit, implementation becomes inexpensive and fast, while the dominant costs migrate to architectural coherence, verification, and cross-boundary alignment. The `lockd` experience offers a concrete, lived confirmation of this hypothesis: a single senior engineer can now span roles that previously required a full team, but doing so demands a sustained, system-level cognitive posture that is qualitatively different from traditional development.

This cognitive posture is best understood as an applied form of systems thinking. The engineer can no longer occupy a single layer of the stack; he must inhabit all layers simultaneously, continuously integrating behaviour across concurrency primitives, storage abstractions, query semantics, operational concerns, and evolving extension points. This aligns closely with foundational work on systems thinking (Senge, 1990; Richmond, 1993; Sterman, 2000) and contemporary definitions of the skill as the ability to reason about interconnected structures, feedback loops, and dynamic complexity (Arnold and Wade, 2015). What is notable is that even a developer with nearly thirty years of experience in distributed systems experiences this as qualitatively new cognitive terrain. Centaur development does not merely magnify traditional expertise; it reconfigures it.

These observations resonate with broader societal forecasts. The World Economic Forum's *Future of Jobs* reports (Forum, 2023; Forum, 2020; Forum, 2018) increasingly identify systems thinking–and its close relative, systems analysis and evaluation–as among the core cognitive skills expected to grow in importance over the coming decade. These reports argue that as technological and socio-technical environments become more complex, human value creation depends less on narrow task execution and more on integrative reasoning across interconnected systems. This prediction mirrors exactly the shift observed in centaur development: as execution accelerates, architectural boundaries, governance mechanisms, and verification strategies become the primary leverage points.

Seen in this light, the centaur developer becomes a micro-scale embodiment of the emerging skill profile anticipated by both organizational theory and labour-market research. The human contribution shifts from manual production to the orchestration of complex systems in partnership with increasingly capable machines. The earlier sections of this article analyze how delivery frameworks and organizational structures might adapt to such autonomous units; this AAR adds the corresponding cognitive dimension. It underscores that what distinguishes effective centaur developers is not the ability to type faster or recall more APIs, but the ability to expand one's system boundary without losing conceptual integrity or ethical responsibility.

The experience with `lockd` thus illustrates both the promise and the strain of this new mode of work. It demonstrates that a centaur unit can achieve architectural and functional breadth once reserved for teams, while it also reveals the cognitive demands of maintaining coherence at machine speed. The result is a development model in which the true "10×" does not lie merely in throughput, but in the ability to design tools, practices, and organizational structures that allow human judgment and machine execution to reinforce—rather than overwhelm—one another. In this sense, the AAR does not simply close the narrative of this essay; it situates centaur development within a broader intellectual and societal trajectory, pointing toward a future where systems thinking becomes not just a recommended skill, but an operational necessity for the practitioners who inhabit the expanding frontier of human–AI collaboration.



REFERENCES

Alves, Ricardo and João Cipriano (2023). “Centaur Intelligence: Human–AI Collaboration Patterns”. In: *arXiv preprint*.

Anderson, David J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*.

Arnold, Ross D. and Jon P. Wade (2015). “A Definition of Systems Thinking: A Systems Approach”. In: *Procedia Computer Science* 44, pp. 669–678. DOI: 10.1016/j.procs.2015.03.050.

Beck, Kent (1999). *Extreme Programming Explained*. Addison-Wesley.

Blomgren, Michel (2025a). *lockd: Distributed Lock Service and Data Platform*. github.com/sa6mwa/lockd.

– (2025b). *Outcome-Based Agile Framework*. github.com/sa6mwa/obaf.

Brooks, Frederick P. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.

Burns, Tom and G. M. Stalker (1961). “The Management of Innovation”. In: *Tavistock Institute*.

Cockburn, Alistair (2001). *Agile Software Development*. Addison-Wesley.

Conway, Melvin E. (1968). “How do Committees Invent?” In: *Datamation* 14.4, pp. 28–31.

Ercin, R. (2025). “Evaluating Code Generation by Large Language Models”. PhD thesis. KTH Royal Institute of Technology. URL: https://www.diva-portal.org/smash/get/diva2:1985844/FULLTEXT01.pdf.

Forsgren, Nicole, Jez Humble, and Gene Kim (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution.

Forum, World Economic (2018). *The Future of Jobs Report 2018*. Earlier emphasis on systems skills within complex problem-solving categories. World Economic Forum. URL: https://www.weforum.org/reports/the-future-of-jobs-report-2018/.

– (2020). *The Future of Jobs Report 2020*. Lists “Systems Analysis and Evaluation” among top skills for 2025. World Economic Forum. URL: https://www.weforum.org/reports/the-future-of-jobs-report-2020/.

– (2023). *The Future of Jobs Report 2023*. Highlights systems thinking and analytical thinking as top growing skills. World Economic Forum. URL: https://www.weforum.org/reports/the-future-of-jobs-report-2023/.

Galbraith, Jay R. (1974). “Organization Design: An Information Processing View”. In: *Interfaces* 4.3, pp. 28–36.

Hamza, Muhammad et al. (2023). “Human AI Collaboration in Software Engineering: Lessons Learned from a Hands On Workshop”. In: *arXiv preprint arXiv:2312.10620*. Accessed November 2025.

Highsmith, Jim (2001). *Agile Software Development Ecosystems*. Addison-Wesley.

Humble, Jez, Joanne Molesky, and Barry O’Reilly (2015). *Lean Enterprise*. O’Reilly Media.

Hutchins, Edwin (1995). *Cognition in the Wild*. MIT Press.

Kasparov, Garry (2017). *The Centaur Model: Advanced Chess and Human–Machine Collaboration*. Interview and essays.

Klein, Gary (2008). *Sources of Power: How People Make Decisions*. MIT Press.

Larman, Craig and Bas Vodde (2017). *Large-Scale Scrum: More with LeSS*. Addison-Wesley.

Mintzberg, Henry (1979). *The Structuring of Organizations*. Prentice Hall.

Office of Government Commerce (2009). *Managing Successful Projects with PRINCE2*.

Oladele, Sunday and Faruk Lawal (2025). “The Impact of AI-Assisted Code Generation on Software Vulnerabilities”. In: *arXiv preprint*.

Peng, Michael et al (2023). “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot”. In: *arXiv preprint arXiv:2302.06590*.

Richmond, Barry (1993). “Systems Thinking: Critical Thinking Skills for the 1990s and Beyond”. In: *System Dynamics Review* 9.2, pp. 113–133. DOI: 10.1002/sdr.4260090203.

Ries, Eric (2011). *The Lean Startup*. Crown Business.

Royce, Winston W. (1970). “Managing the Development of Large Software Systems”. In: *Proceedings of IEEE WESCON*, pp. 1–9.

Sanchez, Ron and Joseph T. Mahoney (1996). “Modularity, Flexibility, and Knowledge Management in Product and Organization Design”. In: *Strategic Management Journal* 17, pp. 63–76.

Scaled Agile Inc. (2021). *SAFe 5.1 Reference Guide*.
Schmutz, Jan B. et al (2024). "AI-Teaming: Redefining Collaboration in the Digital Era". In: *Current Opinion in Psychology*.
Schreiber, M. et al (2025). "Security Vulnerabilities in AI-Generated Code: A Large-Scale Empirical Study". In: *International Conference on Software Security*.
Schwaber, Ken and Jeff Sutherland (2017). *The Scrum Guide*. URL: https://scrumguides.org.
Seeber, Isabella et al (2020). "Machines as Teammates: A Research Agenda on AI in Team Collaboration". In: *Information and Management*. DOI: 10.1016/j.im.2020.103331.
Senge, Peter M. (1990). *The Fifth Discipline: The Art & Practice of the Learning Organization*. Doubleday.
Shneiderman, Ben (2020). "Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy". In: *Communications of the ACM* 63.12, pp. 32–35.
Skelton, Matthew and Manuel Pais (2019). *Team Topologies*. IT Revolution.
Sterman, John D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Boston: Irwin McGraw-Hill.