Abstract

Target length: ~250 words **Status:** Draft v0.1

Abstract

The convergence of multi-agent systems, swarm intelligence, and large language model-based agentic AI creates new possibilities for augmenting systems engineering practice. This survey provides systematic organization of this emerging field, synthesizing contributions across artificial intelligence, software engineering, systems engineering, and human factors research communities.

We present taxonomies of agent architectures—from classical reactive and BDI designs through contemporary LLM-based agents with tool use and memory capabilities—and coordination mechanisms spanning communication-based, organization-based, and emergent approaches. We map applications to ISO/IEC/IEEE 15288 technical processes, revealing concentrated maturity in requirements analysis and test generation with substantial gaps in other lifecycle phases.

Our analysis of evaluation methods identifies critical gaps: no standardized benchmarks exist for multi-agent systems in SE contexts, limiting rigorous cross-study comparison. We survey tools and frameworks, noting rapid maturation of LLM-based agent platforms while observing limited SE-specific integration.

We categorize challenges as technical (scalability, reliability, domain knowledge), integration (tool, process, data), human factors (trust, oversight, skills), and organizational (governance, certification, accountability). Based on challenge analysis, we propose a prioritized research agenda: near-term focus on benchmark development, reliability characterization, and domain knowledge grounding; medium-term attention to coordination at scale, human-AI teaming, and governance frameworks; long-term investigation of collective engineering intelligence and appropriate automation boundaries.

This survey provides researchers organized access to fragmented literature and identifies high-priority gaps. For practitioners, it offers realistic assessment of current capabilities. For the broader community, it establishes common vocabulary and frameworks enabling cumulative progress toward AI-augmented systems engineering.

Keywords: multi-agent systems, swarm intelligence, large language models, agentic AI, systems engineering, ISO 15288, human-AI collaboration

---

Word count: ~280 words **Keywords:** 7

Revision Notes

- [ ] Verify word count meets venue requirements
- [ ] Ensure abstract covers all major sections
- [ ] May need to trim if over limit

Section 1: Introduction

Target length: ~500 words **Status:** Draft v0.1

---

1. Introduction

1.1 Motivation for Survey

The convergence of multi-agent systems research, swarm intelligence, and large language model-based agentic AI creates unprecedented opportunities for augmenting systems engineering practice. Individual AI assistants already help engineers with specific tasks; coordinated multi-agent systems promise more comprehensive support spanning the engineering lifecycle.

Yet this emerging field lacks the systematic organization that mature research areas possess. Literature spans multiple communities—artificial intelligence, software engineering, systems engineering, human factors—with limited cross-referencing. Terminology varies; comparable approaches receive different names; gaps and opportunities remain uncharted. Researchers entering the field face difficulty understanding the landscape; practitioners struggle to assess applicability to their contexts.

This survey addresses these gaps by providing systematic organization of the field, synthesizing contributions across research communities, developing taxonomies of architectures and coordination mechanisms, mapping applications to systems engineering processes, and identifying challenges and research directions.

1.2 Scope and Boundaries

This survey covers multi-agent AI systems applicable to systems engineering—the domain-agnostic discipline addressing complex system development across the lifecycle [24, 25, 26]. We distinguish systems engineering (SE) from the narrower software engineering, though we include software engineering literature where relevant to broader SE application.

Included:

- Multi-agent systems architectures (classical and LLM-based)
- Coordination mechanisms for agent collaboration
- Applications to SE technical processes per ISO 15288
- Evaluation methods and benchmarks
- Tools and frameworks
- Challenges and research directions

Excluded:

- Single-agent AI assistants (covered in related surveys)
- General multi-agent systems without engineering application
- Domain-specific applications without generalizable insights
- Detailed treatment of underlying LLM architectures

1.3 Survey Methodology

This survey follows established systematic review practices adapted for a rapidly evolving field:

Literature identification: We searched IEEE Xplore, ACM Digital Library, Springer, Elsevier, and arXiv using terms including "multi-agent systems," "swarm intelligence," "LLM agents," combined with "systems engineering," "requirements," "architecture," "verification," and related terms. We supplemented searches with citation following and expert recommendation.

Selection criteria: We included peer-reviewed publications and significant preprints addressing multi-agent AI approaches with relevance to systems engineering processes. We excluded purely theoretical work without application consideration and purely domain-specific applications without generalizable contribution.

Synthesis approach: We organized findings according to taxonomic frameworks developed iteratively during review. We assessed application maturity based on evidence type (laboratory, field trial, production) and validation rigor.

Limitations: The field evolves rapidly; some recent work may be missed. LLM-based agent research moves faster than peer review; we include significant preprints while noting lower validation standards. Our SE perspective may underweight

contributions from adjacent fields.

1.4 Paper Organization

The remainder of this paper is organized as follows:

- Section 2 provides background on multi-agent systems, swarm intelligence, LLM-based agents, and systems engineering processes
- Section 3 presents a taxonomy of agent architectures
- Section 4 surveys coordination mechanisms
- Section 5 maps applications to systems engineering process areas
- Section 6 examines evaluation methods and benchmarks
- Section 7 reviews tools and frameworks
- Section 8 analyzes challenges and open problems
- Section 9 proposes research directions
- Section 10 concludes

Word count: ~520 words **Subsections:** 4

Revision Notes

- [ ] Update literature search details with actual search results
- [ ] Add figure showing survey methodology
- [ ] Consider adding reading guide for different audiences
- [ ] Verify section numbering matches final organization

Section 2: Background and Foundations

Target length: ~1,200 words **Status:** Draft v0.1

---

2. Background and Foundations

This section establishes the foundational concepts underlying multi-agent AI systems for systems engineering, tracing the evolution of relevant research areas and defining the intersection that motivates this survey.

2.1 Multi-Agent Systems: Historical Development

Multi-agent systems (MAS) emerged as a distinct research area in the 1980s and 1990s, drawing from distributed artificial intelligence, object-oriented programming, and concurrent systems [1, 2]. Wooldridge and Jennings [3] provided influential definitions, characterizing agents as computer systems capable of autonomous action in some environment to meet design objectives. Key properties include autonomy (operating without direct intervention), social ability (interacting with other agents), reactivity (perceiving and responding to environment), and proactivity (taking initiative toward goals).

Early MAS research addressed fundamental questions: How should agents be designed internally? How should agents communicate and coordinate? How do collective behaviors emerge from individual actions? Foundational work established agent communication languages (KQML, FIPA-ACL), coordination mechanisms (contract net, organizational structures), and agent architectures (reactive, deliberative, hybrid) [4, 5, 6].

By the 2000s, MAS had found applications across domains including manufacturing, logistics, e-commerce, and simulation. However, the symbolic AI approaches underlying most MAS work faced limitations in handling uncertainty, learning from experience, and processing unstructured information—limitations that constrained applicability to complex real-world problems.

2.2 Swarm Intelligence: Principles and Paradigms

Swarm intelligence emerged from study of collective behavior in biological systems—ant colonies, bird flocks, fish schools—where simple individuals following local rules produce sophisticated collective behavior [7, 8]. Bonabeau, Dorigo, and Théraulaz [9] formalized swarm intelligence principles and demonstrated applications to optimization problems.

Key swarm intelligence paradigms include:

Ant Colony Optimization (ACO) models how ants find shortest paths through pheromone-based stigmergic coordination [10]. Artificial ants deposit virtual pheromone on solution components, guiding subsequent ants toward promising solutions. ACO has proven effective for combinatorial optimization problems including routing, scheduling, and assignment.

Particle Swarm Optimization (PSO) models flocking behavior, with particles (candidate solutions) moving through search spaces influenced by their own best-known position and the swarm's best-known position [11]. PSO has found wide application in continuous optimization.

Stigmergic coordination provides a paradigm for indirect communication through environmental modification—agents leave traces that influence other agents' behavior without direct message exchange [12]. This enables scalable coordination with minimal communication overhead.

Swarm intelligence contributes key insights to AI swarms: that collective intelligence can exceed individual capability, that simple local rules can produce complex global behavior, and that decentralized coordination can achieve robust, scalable solutions.

2.3 Large Language Models and Agentic AI

The emergence of large language models (LLMs) since 2018 has transformed the landscape for AI agent development [13, 14]. Models trained on massive text corpora exhibit emergent capabilities in language understanding, reasoning, and generation that earlier approaches could not achieve [15].

The transition from LLMs to agentic AI involves augmenting language models with capabilities for autonomous action:

Tool use enables LLMs to invoke external tools—calculators, search engines, APIs, code interpreters—extending their capabilities beyond text generation [16, 17]. Tool-augmented LLMs can retrieve information, execute computations, and interact with external systems.

Memory systems provide LLMs with persistent context beyond their context window, enabling accumulation of knowledge across interactions [18]. Memory architectures include retrieval-augmented generation (RAG), vector databases, and structured knowledge stores.

Planning and reasoning approaches enable LLMs to decompose complex tasks, reason about action sequences, and pursue goals over extended interactions [19, 20]. Techniques include chain-of-thought prompting, ReAct (reasoning + acting), and tree-of-thought exploration.

Multi-agent LLM systems extend agentic AI to multiple coordinated agents, each potentially with specialized roles, capabilities, or perspectives [21, 22, 23]. Recent frameworks including AutoGPT, MetaGPT, CrewAI, and others demonstrate multi-agent architectures where LLM-based agents collaborate on complex tasks.

2.4 Systems Engineering Process Framework

Systems engineering provides the disciplined processes for managing complex system development across the lifecycle [24, 25]. ISO/IEC/IEEE 15288:2023 [26] defines the standard framework of system lifecycle processes, organized into:

Technical processes directly realize and support the system:

- Stakeholder Needs and Requirements Definition
- System Requirements Definition
- Architecture Definition
- Design Definition
- System Analysis
- Implementation
- Integration
- Verification
- Validation
- Transition
- Operation
- Maintenance
- Disposal

Technical management processes establish and evolve plans, assess progress, and control execution. **Organizational project-enabling processes** provide resources and infrastructure.

The INCOSE Systems Engineering Handbook [27] and NASA Systems Engineering Handbook [28] provide guidance on applying these processes across domains. Systems engineering applies to any complex system—aerospace, defense, healthcare, energy, transportation, infrastructure—making it inherently domain-agnostic while requiring domain-specific knowledge for application.

2.5 Intersection: Why Multi-Agent AI for Systems Engineering?

The convergence of multi-agent systems, swarm intelligence, and LLM-based agentic AI creates new possibilities for systems engineering support. Several characteristics of systems engineering motivate multi-agent approaches:

Multi-disciplinary nature. Systems engineering integrates contributions from diverse disciplines—mechanical, electrical, software, human factors, logistics. No single agent can embody expertise across all disciplines; multi-agent architectures enable discipline specialization with cross-discipline coordination.

Collaborative practice. Systems engineering is inherently collaborative, involving teams of engineers, stakeholders, and organizations. Multi-agent systems mirror this collaborative structure, with agents representing different perspectives, roles, or organizational entities.

Artifact complexity. Systems engineering produces complex, interrelated artifacts—requirements, architectures, designs, test cases—with consistency and traceability requirements across thousands of elements. Agent swarms can achieve comprehensive coverage that individual analysis cannot.

Lifecycle span. Systems engineering spans extended lifecycles from concept through disposal. Persistent agent systems can maintain continuity, accumulating knowledge and adapting to evolving system states.

Quality through diversity. Engineering quality benefits from multiple perspectives examining artifacts. Agent swarms provide perspective diversity—different specializations, different analysis approaches—that surface issues single-point analysis misses.

This intersection defines the scope of this survey: multi-agent AI systems, drawing on classical MAS foundations, swarm intelligence principles, and modern LLM-based agentic AI capabilities, applied to support systems engineering processes.

Word count: ~1,050 words **Subsections:** 5 **References cited:** [1]-[28]

Revision Notes

- [ ] Expand LLM-based agents section with more recent frameworks
- [ ] Add specific examples for each swarm intelligence paradigm
- [ ] Consider adding timeline figure

Section 3: Taxonomy of Agent Architectures

Target length: ~1,500 words **Status:** Draft v0.1

---

3. Taxonomy of Agent Architectures

This section presents a taxonomy of agent architectures relevant to systems engineering applications, tracing evolution from classical approaches through contemporary LLM-based designs.

3.1 Classification Framework

Agent architectures can be classified along several dimensions:

Reasoning approach: How does the agent decide what to do?

- Reactive: Stimulus-response mappings
- Deliberative: Explicit reasoning about goals and plans
- Hybrid: Combining reactive and deliberative elements

Knowledge representation: How does the agent represent its understanding?

- Implicit: Encoded in behavior rules or neural weights
- Explicit: Symbolic knowledge bases, ontologies
- Hybrid: Combining implicit and explicit representations

Learning capability: Can the agent improve from experience?

- Static: Fixed behavior after deployment
- Adaptive: Learning during operation
- Meta-learning: Learning how to learn

Foundation model: What provides core capabilities?

- Classical AI: Rule-based, search, optimization
- Machine learning: Statistical models, neural networks
- Large language models: Pre-trained transformer models

3.2 Reactive Architectures

Reactive architectures eschew explicit world models and deliberation in favor of direct stimulus-response mappings. Brooks' subsumption architecture [29] demonstrated that intelligent behavior could emerge from layered reactive behaviors without symbolic reasoning.

Characteristics:

- Fast response to environmental changes

- Robust to model errors (no model to be wrong)
- Limited planning horizon
- Behavior emerges from interaction of simple rules

SE applications: Reactive architectures suit real-time monitoring and response—detecting anomalies, triggering alerts, managing routine workflows. They are less suited to complex reasoning tasks like trade studies or requirements analysis.

Limitations for SE: Systems engineering frequently requires reasoning about abstract concepts (requirements, architectures) that lack direct environmental correlates. Pure reactive approaches cannot address tasks requiring deliberation, planning, or explanation.

3.3 Deliberative Architectures

Deliberative architectures maintain explicit world models and reason about goals, plans, and actions. The Belief-Desire-Intention (BDI) model [30, 31] provides an influential framework where agents have beliefs about the world, desires (goals) they wish to achieve, and intentions (committed plans) they are executing.

Characteristics:

- Explicit goal representation and pursuit
- Planning and plan execution
- Reasoning about action consequences
- Explainable decision-making (in principle)

SE applications: BDI agents can model engineering roles with explicit goals (e.g., a requirements agent with goals of completeness, consistency, traceability) and plans for achieving them (analysis procedures, review protocols). Goal-directed behavior aligns with engineering objectives.

Implementations: JACK, Jadex, Jason provide BDI agent platforms [32, 33]. These frameworks support agent development with explicit belief bases, goal management, and plan libraries.

Limitations for SE: Classical BDI implementations require explicit knowledge engineering—defining beliefs, goals, and plans in advance. This limits adaptability to novel situations and requires substantial development effort to encode domain knowledge.

3.4 Hybrid Architectures

Hybrid architectures combine reactive and deliberative elements, typically in layered designs where reactive behaviors handle routine situations while deliberative reasoning addresses complex cases.

Layered architectures [34] organize agent capabilities hierarchically:

- Reactive layer: Immediate responses to environmental conditions
- Planning layer: Goal-directed reasoning and plan generation
- Cooperative layer: Social reasoning and coordination with other agents

InteRRaP [35] exemplifies layered hybrid design with behavior-based, planning, and cooperation layers, each with associated knowledge bases.

SE applications: Hybrid architectures can provide responsive behavior for routine engineering tasks while engaging deliberative reasoning for complex decisions. A requirements agent might reactively maintain traceability links while deliberatively analyzing completeness.

3.5 LLM-Based Agent Architectures

Large language model-based agents represent a paradigm shift, leveraging pre-trained foundation models rather than hand-crafted knowledge bases. Several architectural patterns have emerged:

ReAct (Reasoning + Acting) [36] interleaves reasoning traces with actions, enabling LLMs to reason about what action to take, execute it, observe results, and continue reasoning. ReAct provides a general pattern for LLM-based agency.

Tool-augmented architectures [37, 38] extend LLM capabilities through tool invocation. The LLM decides which tools to call, formulates inputs, interprets outputs, and integrates results into ongoing reasoning. Tools can include calculators, search engines, code interpreters, and domain-specific applications.

Memory-augmented architectures [39] provide persistent memory beyond the LLM's context window:

- Short-term memory: Recent interaction context
- Long-term memory: Persistent knowledge, accumulated experience
- Episodic memory: Records of past interactions and outcomes

Retrieval-Augmented Generation (RAG) [40] grounds LLM responses in retrieved documents, enabling access to domain knowledge, standards, and organizational practices without fine-tuning.

Multi-agent LLM architectures coordinate multiple LLM-based agents:

- Role-based: Agents assume specialized roles (researcher, coder, reviewer)
- Debate-based: Agents argue positions, refine through discourse
- Hierarchical: Manager agents coordinate worker agents
- Peer: Agents collaborate as equals with complementary capabilities

SE-relevant frameworks:

- **MetaGPT** [41]: Multi-agent software development with role-based specialization
- **AutoGen** [42]: Conversational agents with customizable patterns
- **CrewAI** [43]: Role-based agent crews for complex tasks
- **LangGraph** [44]: Graph-based agent orchestration

3.6 Architecture Comparison

Table 1 compares architectural approaches across key dimensions.

Architecture	Reasoning	Knowledge	Learning	Planning	Explainability	SE Fit
Reactive	Stimulus-response	Implicit	Limited	None	Low	Monitoring
BDI	Goal-directed	Explicit	Limited	Yes	High	Structured tasks
Hybrid	Layered	Mixed	Moderate	Partial	Moderate	Balanced tasks
LLM-based	Emergent	Implicit + RAG	In-context	Via prompting	Moderate	Flexible tasks
LLM + Tools	Augmented	Hybrid	Adaptive	Via reasoning	Moderate	Complex tasks

Multi-agent LLM	Distributed	Distributed	Collective	Coordinated	Variable	Full lifecycle

3.7 Architectural Trends

Several trends characterize the evolution of agent architectures for engineering applications:

Foundation model dominance. LLM-based architectures increasingly dominate new development, leveraging pre-trained capabilities rather than hand-crafted knowledge. This shifts development effort from knowledge engineering to prompt engineering and tool integration.

Hybrid knowledge integration. Pure LLM approaches suffer from hallucination and knowledge gaps; production systems increasingly integrate retrieval (RAG), structured knowledge, and tool-based verification to ground LLM reasoning.

Specialization through prompting. Rather than architecturally distinct agent types, LLM-based systems achieve specialization through role prompts, system instructions, and tool configurations applied to a common foundation model.

Emergent coordination. Multi-agent LLM systems exhibit coordination patterns not explicitly programmed, emerging from agent interactions. This creates both opportunities (novel solutions) and challenges (unpredictable behavior).

Continuous evolution. Unlike classical agents with fixed architectures, LLM-based agents evolve as foundation models improve. Architectural patterns that leverage model capabilities will benefit from ongoing model advancement.

Word count: ~1,100 words **Subsections:** 7 **Tables:** 1 **References cited:** [29]-[44]

Revision Notes

- [ ] Add more detail on specific LLM agent frameworks
- [ ] Consider adding architecture diagram
- [ ] Expand comparison table with more dimensions
- [ ] Add citations for recent multi-agent frameworks

Section 4: Coordination Mechanisms

Target length: ~1,200 words **Status:** Draft v0.1

---

4. Coordination Mechanisms

Effective multi-agent systems require coordination—managing dependencies, resolving conflicts, and achieving coherent collective behavior. This section surveys coordination mechanisms applicable to AI swarms in systems engineering contexts.

4.1 Classification of Coordination Approaches

Malone and Crowston [45] define coordination as "managing dependencies between activities." Coordination mechanisms can be classified by:

Communication directness:

- Direct: Explicit message exchange between agents

- Indirect: Coordination through shared environment (stigmergy)

Control structure:

- Centralized: Coordinator manages agent activities
- Decentralized: Agents coordinate peer-to-peer
- Hierarchical: Multi-level coordination structure

Temporal coupling:

- Synchronous: Agents coordinate in real-time
- Asynchronous: Agents coordinate through persistent state

Formality:

- Structured: Defined protocols and interfaces
- Unstructured: Natural language or emergent patterns

4.2 Communication-Based Coordination

Communication-based approaches coordinate through explicit information exchange:

Message passing provides direct agent-to-agent communication. Agent communication languages (ACL) define message semantics [46]. FIPA-ACL specifies performatives (inform, request, propose) enabling structured dialogue. In LLM-based systems, natural language messages often replace formal ACLs.

Blackboard systems [47] coordinate through shared data structures. Agents post contributions to a shared "blackboard"; other agents observe and respond. This decouples agents temporally and reduces direct communication complexity. For SE, shared system models can serve as blackboards—agents contribute analyses, other agents incorporate findings.

Publish-subscribe patterns enable event-driven coordination [48]. Agents publish events; interested agents subscribe and receive notifications. This supports loose coupling and scalability. SE applications include change propagation—when requirements change, subscribed architecture agents receive notifications.

Shared memory provides coordination through common state access. Agents read and write shared data structures, observing each other's contributions. Vector databases in LLM systems serve as shared memory, enabling agents to store and retrieve information produced by other agents.

4.3 Organization-Based Coordination

Organization-based approaches structure agent relationships to manage coordination:

Hierarchical organizations arrange agents in authority structures [49]. Higher-level agents decompose tasks, assign work, and integrate results. This reduces coordination complexity—each agent coordinates primarily with its supervisor and subordinates. SE parallels: chief systems engineer coordinating IPT leads who coordinate discipline engineers.

Market-based coordination uses economic mechanisms [50]. The Contract Net Protocol [51] exemplifies market coordination: manager agents announce tasks; contractor agents bid; managers award contracts based on bids. This enables dynamic task allocation without centralized planning.

Team-based coordination groups agents into teams pursuing shared goals [52]. Teamwork theories (SharedPlans, Joint Intentions) formalize how agents commit to collective objectives and coordinate execution. SE teams naturally map to agent teams with shared project goals.

Coalition formation enables dynamic grouping for specific objectives [53]. Agents form coalitions when collaboration benefits exceed costs, dissolving when objectives are met. For SE, coalitions might form for specific trade studies or reviews, then

dissolve.

4.4 Emergent Coordination

Emergent approaches achieve coordination without explicit protocols:

Stigmergy coordinates through environmental modification [12]. Agents leave traces (pheromones, markers) that influence other agents' behavior. No direct communication required; coordination emerges from accumulated environmental changes. SE applications: agents annotating shared models, with annotations guiding subsequent agent attention.

Self-organization produces coordinated structures from local interactions [54]. Agents following simple local rules generate global patterns without central control. This enables robust, scalable coordination but limits predictability.

Swarm behaviors exhibit collective intelligence through decentralized coordination [9]. Individual agents respond to local information; collective behavior emerges from aggregated responses. Swarm coordination suits exploration and optimization but may struggle with precision requirements.

4.5 Hybrid Approaches

Practical systems often combine coordination mechanisms:

Hierarchical with market elements uses hierarchy for task decomposition and markets for resource allocation within levels.

Centralized planning with distributed execution employs central coordinators for planning while agents execute autonomously, reporting status.

Structured protocols with natural language combines formal coordination protocols with natural language communication for flexibility.

4.6 Coordination in LLM-Based Systems

LLM-based multi-agent systems exhibit distinctive coordination patterns:

Conversational coordination uses natural language dialogue rather than formal protocols. Agents discuss, debate, and negotiate in natural language, leveraging LLM conversational capabilities. This enables flexibility but may sacrifice precision.

Prompt-based role assignment coordinates through role definitions in system prompts. Each agent's prompt defines its responsibilities, communication patterns, and coordination expectations.

Orchestrator patterns employ a central agent (often called "supervisor" or "manager") that routes tasks, manages workflow, and synthesizes outputs. This provides control but creates bottlenecks.

Reflection and critique patterns have agents review each other's outputs, providing feedback that drives iteration. Coordination emerges through iterative refinement.

Memory-mediated coordination uses shared memory systems (vector stores, knowledge graphs) as coordination substrate. Agents contribute to and query shared memory, achieving indirect coordination.

4.7 Comparison and Trade-offs

Table 2 compares coordination mechanisms across key dimensions.

Mechanism	Coupling	Scalability	Predictability	Flexibility	SE Suitability
Message passing	Tight	Moderate	High	Moderate	Structured workflows

Blackboard	Loose	Good	Moderate	High	Shared model updates
Hierarchy	Moderate	Good	High	Low	Large teams
Market	Loose	Good	Moderate	High	Resource allocation
Stigmergy	Loose	Excellent	Low	High	Exploration tasks
Conversational	Tight	Limited	Low	Excellent	Creative tasks
Orchestrator	Tight	Limited	High	Moderate	Controlled workflows

Trade-off considerations for SE:

Predictability vs. flexibility: Safety-critical SE applications prioritize predictability; exploratory tasks benefit from flexibility. Coordination mechanism choice should align with task requirements.

Scalability vs. control: Larger swarms require scalable coordination (hierarchical, stigmergic) but may sacrifice fine-grained control. Task characteristics determine appropriate balance.

Communication overhead: Rich communication (conversational) enables nuanced coordination but incurs token costs and latency. Efficient protocols reduce overhead at the cost of expressiveness.

Emergence management: Emergent coordination can produce beneficial novelty or harmful unexpected behavior. SE contexts may require bounds on emergence through hybrid approaches.

Word count: ~980 words **Subsections:** 7 **Tables:** 1 **References cited:** [45]-[54]

Revision Notes

- [ ] Add specific examples from LLM multi-agent frameworks
- [ ] Consider adding coordination diagram
- [ ] Expand trade-off analysis with quantitative considerations
- [ ] Add discussion of coordination failures and mitigations

Section 5: Applications to Systems Engineering

Target length: ~2,000 words **Status:** Draft v0.1

---

5. Applications to Systems Engineering

This section surveys multi-agent AI applications across systems engineering process areas, assessing maturity and identifying gaps in current research and practice.

5.1 Mapping Framework

We organize applications according to ISO/IEC/IEEE 15288:2023 technical processes [26], providing a systematic framework for coverage assessment. For each process area, we examine:

- Reported applications in literature

- Agent architectures employed
- Maturity level (research prototype, pilot deployment, production use)
- Evidence quality (laboratory study, field trial, case study)

5.2 Requirements Engineering Applications

Requirements engineering—encompassing stakeholder needs definition and system requirements definition—has received substantial attention from AI and multi-agent researchers.

Requirements elicitation applications use agents to support stakeholder interaction and needs capture. Multi-agent approaches include:

- Stakeholder-representative agents embodying different user perspectives [55]
- Interview assistant agents generating questions and probing for completeness
- Multi-perspective analysis where agents identify conflicting stakeholder needs

Requirements analysis applications examine requirements quality:

- Completeness checking agents identifying missing requirements based on domain templates [56]
- Consistency checking agents detecting conflicts between requirements [57]
- Ambiguity detection agents flagging unclear language [58]
- Multi-agent argumentation for resolving requirement conflicts [59]

Requirements specification applications generate requirements artifacts:

- Natural language generation of requirements from models
- Formalization agents translating natural language to formal specifications
- Template-based generation with multi-agent review

Requirements traceability applications maintain relationships:

- Trace link generation agents identifying relationships between artifacts [60]
- Impact analysis agents assessing change propagation
- Traceability verification agents checking link completeness

Maturity assessment: Requirements applications are among the most mature, with research spanning decades and recent LLM-based tools reaching pilot deployment. Evidence includes controlled experiments and industrial case studies.

5.3 Architecture and Design Applications

Architecture definition and design processes have seen growing multi-agent application:

Architecture exploration employs agents for systematic alternative evaluation:

- Multi-objective optimization agents exploring architecture trade spaces [61]
- Pattern-matching agents suggesting applicable architecture patterns
- Constraint-checking agents verifying architecture decisions against requirements

View generation applications produce architecture representations:

- Viewpoint-specialized agents generating views per ISO 42010 [62]
- Diagram generation agents creating visual representations
- Consistency-checking agents verifying cross-view alignment

Interface definition applications manage system boundaries:

- Interface identification agents detecting needed interfaces from architectures
- Interface specification agents generating ICD content
- Conflict detection agents identifying interface mismatches

Trade study support coordinates multi-criteria analysis:

- Alternative generation agents proposing design options
- Evaluation agents assessing alternatives against criteria
- Sensitivity analysis agents exploring parameter variations
- Multi-agent deliberation aggregating assessments [63]

Maturity assessment: Architecture applications are moderately mature. Optimization-based approaches have production use in specific domains; LLM-based architecture agents remain largely research prototypes with limited industrial validation.

5.4 Verification and Validation Applications

V&V processes have substantial multi-agent application, particularly for software-intensive systems:

Test case generation employs agents for systematic test development:

- Requirements-based test generation agents deriving tests from specifications [64]
- Coverage-directed agents generating tests to maximize coverage criteria
- Adversarial agents generating challenging test cases
- Multi-agent approaches combining coverage, boundary, and fault-based strategies

Test execution and analysis coordinates automated testing:

- Test orchestration agents managing test execution
- Result analysis agents interpreting test outcomes
- Regression analysis agents identifying failure patterns

Formal verification applications support rigorous analysis:

- Model checking agents exploring state spaces
- Theorem proving agents assisting formal proofs
- Abstraction agents managing complexity through model reduction

Review and inspection applications support human review processes:

- Pre-review analysis agents identifying potential issues for human attention
- Checklist verification agents assessing completeness against criteria
- Multi-perspective review agents examining artifacts from different viewpoints [65]

Maturity assessment: V&V applications, particularly test generation, are relatively mature with commercial tools incorporating AI capabilities. Formal verification agents remain research-focused. LLM-based review support is emerging rapidly.

5.5 Integration and Lifecycle Applications

Later lifecycle phases have received less attention but offer significant opportunities:

Integration planning applications support assembly sequencing:

- Dependency analysis agents identifying integration constraints
- Sequence optimization agents generating build orders
- Resource allocation agents managing integration facilities

Transition support applications facilitate deployment:

- Procedure generation agents creating transition plans
- Training material agents generating user documentation
- Readiness assessment agents evaluating transition prerequisites

Operations support applications assist in-service systems:

- Anomaly detection agents identifying unexpected behaviors
- Diagnostic agents supporting fault isolation
- Operational optimization agents suggesting efficiency improvements

Maintenance support applications sustain system capability:

- Failure prediction agents anticipating maintenance needs
- Sustainment planning agents optimizing maintenance schedules
- Technical refresh agents assessing modernization options

Maturity assessment: Lifecycle applications lag requirements and V&V in maturity. Operations and maintenance applications exist for specific domains (predictive maintenance in manufacturing) but general SE applications remain limited.

5.6 Cross-Cutting Applications

Some applications span multiple process areas:

Documentation generation produces engineering artifacts:

- Report generation agents synthesizing technical documents
- Specification writing agents creating standards-compliant documents
- Update propagation agents maintaining document consistency

Traceability management maintains artifact relationships:

- Link discovery agents identifying implicit relationships
- Impact analysis agents propagating changes through trace networks
- Compliance verification agents checking traceability completeness

Configuration management tracks artifact versions and baselines:

- Change detection agents identifying modifications
- Baseline comparison agents assessing evolution
- Configuration audit agents verifying consistency

Project management support assists technical management:

- Progress assessment agents evaluating completion status
- Risk identification agents surfacing technical risks
- Resource estimation agents predicting effort requirements

5.7 Application Maturity Assessment

Table 3 summarizes application maturity across process areas.

Process Area	Research Activity	Tool Availability	Industrial Adoption	Evidence Quality

Requirements elicitation	High	Moderate	Low-Moderate	Case studies
Requirements analysis	High	Moderate	Moderate	Experiments + cases
Architecture exploration	Moderate	Low	Low	Prototypes
Trade studies	Moderate	Low	Low	Laboratory
Test generation	High	High	Moderate	Experiments + field
Formal verification	Moderate	Moderate	Low	Laboratory
Review support	Emerging	Low	Very low	Prototypes
Integration	Low	Very low	Very low	Limited
Operations/maintenance	Moderate	Moderate (domain-specific)	Moderate (specific domains)	Case studies
Documentation	Emerging	Emerging	Very low	Prototypes

5.8 Summary: Applications × Process × Evidence

Key observations from the application survey:

Concentrated maturity: Application maturity concentrates in requirements analysis and test generation, with other process areas substantially less developed.

LLM acceleration: LLM-based approaches are rapidly advancing across process areas, but with limited industrial validation. Most LLM applications remain research prototypes or early pilots.

Domain specificity: Many mature applications are domain-specific (aerospace V&V, manufacturing maintenance) rather than general SE tools.

Multi-agent gap: Most applications employ single agents or simple agent pairs; true multi-agent swarm approaches remain rare except in optimization contexts.

Evidence limitations: Rigorous empirical evaluation is limited. Many papers report prototype demonstrations without controlled experiments or industrial validation.

Word count: ~1,150 words **Subsections:** 8 **Tables:** 1 **References cited:** [55]-[65]

Revision Notes

- [ ] Expand with specific tool and framework names
- [ ] Add more citations for each application area
- [ ] Consider splitting into two sections if length grows
- [ ] Add discussion of cross-application integration

Section 6: Evaluation Methods and Benchmarks

Target length: ~800 words **Status:** Draft v0.1

---

6. Evaluation Methods and Benchmarks

Rigorous evaluation is essential for advancing multi-agent AI systems in systems engineering. This section surveys evaluation approaches, existing benchmarks, and gaps requiring attention.

6.1 Evaluation Challenges for MAS in SE

Evaluating multi-agent systems for systems engineering presents distinctive challenges:

Task complexity. SE tasks involve extended reasoning, multiple artifacts, and judgment calls that resist simple correctness metrics. Unlike classification or generation tasks with clear ground truth, SE tasks often have multiple acceptable solutions.

Interaction effects. Multi-agent system performance depends on agent interactions, not just individual capabilities. Evaluation must capture collective behavior, coordination effectiveness, and emergent properties.

Context dependence. SE effectiveness depends heavily on domain, organizational context, and specific project characteristics. Results from one context may not generalize.

Temporal scope. SE processes span extended periods; full evaluation of lifecycle support requires longitudinal assessment impractical for most research studies.

Human factors. Effectiveness ultimately depends on human engineer productivity, satisfaction, and trust—subjective factors difficult to measure in controlled settings.

6.2 Existing Benchmarks and Datasets

The field lacks standardized benchmarks for multi-agent SE applications. Relevant existing resources include:

Software engineering benchmarks:

- HumanEval, MBPP for code generation [66]
- SWE-bench for repository-level coding tasks [67]
- These address software engineering specifically, not broader SE

Requirements engineering datasets:

- PURE dataset of requirements and domain knowledge [68]
- NFR dataset for requirements classification
- Limited scale and domain coverage

General agent benchmarks:

- AgentBench for LLM agent capabilities [69]
- GAIA for general AI assistants [70]
- WebArena for web-based tasks
- These assess general capabilities, not SE-specific performance

Multi-agent benchmarks:

- ChatArena for multi-agent dialogue
- Limited coverage of collaborative task completion

Gap: No comprehensive benchmark suite exists for multi-agent systems applied to systems engineering processes.

6.3 Metrics Used in Literature

Studies evaluating AI applications in SE employ varied metrics:

Task-specific metrics:

- Requirements completeness (percentage of expected requirements identified)
- Defect detection rate (percentage of seeded defects found)
- Test coverage achieved
- Design space coverage

Quality metrics:

- Accuracy (correctness of outputs)
- Precision/recall for classification tasks
- BLEU/ROUGE for generation tasks (limited applicability to SE)

Efficiency metrics:

- Time to completion
- Token usage / computational cost
- Human effort reduction

Coordination metrics:

- Communication volume
- Convergence time
- Conflict frequency and resolution

Human-centered metrics:

- User satisfaction (survey-based)
- Trust measures
- Cognitive load
- Adoption intention

6.4 Gaps in Evaluation Methods

Critical gaps limit rigorous evaluation:

Benchmark absence. No standard benchmarks enable cross-study comparison of multi-agent SE systems. Each study uses different tasks, datasets, and metrics, preventing cumulative progress assessment.

Realism deficit. Available datasets often lack the complexity, scale, and domain richness of real SE problems. Toy problems may not predict performance on realistic tasks.

Coordination evaluation. Methods for assessing multi-agent coordination quality—beyond simple task completion—remain underdeveloped. How do we measure whether coordination was efficient, robust, or appropriate?

Longitudinal methods. Evaluating lifecycle support requires methods for assessing systems over extended periods and across multiple project phases—rarely practical in research settings.

Human-AI teaming evaluation. Methods for assessing human-AI collaboration effectiveness—not just AI capability in isolation—need development. How do we measure whether the human-AI team performs better than human-only or AI-only alternatives?

6.5 Toward SE-Specific Benchmarks

Addressing evaluation gaps requires community investment in benchmark development:

Task suite development. Creating standardized task suites for each SE process area, with realistic complexity, domain diversity, and ground truth or expert reference solutions.

Evaluation protocol standardization. Establishing common evaluation protocols enabling fair comparison across systems, including standard train/test splits, evaluation procedures, and reporting requirements.

Multi-dimensional metrics. Developing metrics capturing multiple performance dimensions—correctness, efficiency, coordination quality, human compatibility—rather than single-number summaries.

Living benchmarks. Creating benchmarks that evolve as capabilities advance, preventing saturation while maintaining comparability through versioning.

Community infrastructure. Building shared infrastructure for benchmark hosting, evaluation automation, and result aggregation enabling efficient community-wide evaluation.

Benchmark development requires collaboration between AI researchers and SE practitioners to ensure benchmarks reflect realistic engineering challenges while remaining tractable for research evaluation.

Word count: ~720 words **Subsections:** 5 **References cited:** [66]-[70]

Revision Notes

- [ ] Expand list of existing benchmarks
- [ ] Add specific examples of metrics from key papers
- [ ] Consider proposing specific benchmark characteristics
- [ ] Add discussion of benchmark development challenges

Section 7: Tools and Frameworks

Target length: ~800 words **Status:** Draft v0.1

---

7. Tools and Frameworks

This section surveys tools and frameworks enabling multi-agent AI system development, from classical MAS platforms through contemporary LLM-based frameworks, and examines integration with systems engineering tools.

7.1 Multi-Agent Platforms

Classical multi-agent platforms provide infrastructure for agent development and deployment:

JADE (Java Agent DEvelopment Framework) [71] provides FIPA-compliant agent infrastructure including agent lifecycle management, communication, and directory services. Widely used in research and education, JADE offers mature infrastructure

but requires substantial development effort for sophisticated agents.

SPADE (Smart Python Agent Development Environment) [72] offers Python-based agent development with XMPP-based communication. Its Python foundation facilitates integration with modern AI libraries.

Jason [73] implements the AgentSpeak language for BDI agent development, providing declarative agent programming with explicit beliefs, goals, and plans. Jason suits applications requiring explainable agent reasoning.

MASON [74] provides a fast discrete-event multi-agent simulation library in Java, suited for large-scale agent-based modeling and simulation.

Mesa [75] offers Python-based agent-based modeling, providing accessible ABM capabilities with visualization support.

Limitations for SE: Classical platforms require explicit agent programming and lack native integration with LLM capabilities. They provide infrastructure but not the reasoning capabilities modern SE applications require.

7.2 LLM-Based Agent Frameworks

The emergence of LLMs has spawned frameworks specifically supporting LLM-based agents:

LangChain [76] provides abstractions for LLM application development including chains, agents, and tools. LangChain agents can use tools, maintain memory, and execute multi-step reasoning. Widely adopted with extensive tool ecosystem.

LlamaIndex [77] focuses on connecting LLMs with external data through indexing and retrieval, enabling RAG-based applications. Strong support for document processing and knowledge integration.

AutoGPT [78] pioneered autonomous LLM agents pursuing goals through iterative reasoning and action. Demonstrated potential for autonomous task completion but also limitations in reliability and control.

MetaGPT [79] implements multi-agent software development with specialized roles (product manager, architect, engineer). Demonstrates role-based multi-agent patterns for software engineering tasks.

CrewAI [80] provides framework for orchestrating role-based agent "crews" with defined roles, goals, and collaboration patterns. Emphasizes agent specialization and coordination.

AutoGen [81] from Microsoft supports conversational agents with customizable interaction patterns, enabling flexible multi-agent dialogue configurations.

LangGraph [82] extends LangChain with graph-based agent orchestration, supporting complex multi-agent workflows with explicit state management and control flow.

Comparison considerations:

- LangChain/LlamaIndex: General-purpose, extensive ecosystem
- MetaGPT/CrewAI: Role-based specialization, team metaphors
- AutoGen: Conversational patterns, research-oriented
- LangGraph: Workflow orchestration, state management

7.3 SE Tool Integration

Effective multi-agent SE applications require integration with existing engineering tools:

MBSE tools:

- Cameo Systems Modeler / MagicDraw (SysML modeling)
- IBM Engineering Systems Design Rhapsody

- Capella (MBSE for systems architects)
- Integration typically via APIs, model import/export, or direct database access

Requirements management:

- IBM DOORS / DOORS Next
- Jama Connect
- Polarion
- Integration enables AI agents to read, analyze, and potentially update requirements

PLM/PDM systems:

- Siemens Teamcenter
- PTC Windchill
- Dassault 3DEXPERIENCE
- Integration provides access to product data, configurations, and workflows

ALM tools:

- Jira (issue/task tracking)
- Azure DevOps
- GitLab
- Integration supports workflow coordination and artifact management

Integration challenges:

- Proprietary APIs and data formats
- Authentication and access control
- Data consistency and transaction management
- Performance at scale

7.4 Capability Comparison

Table 4 compares framework capabilities relevant to SE applications.

Framework	Multi-Agent	Memory	Tools	Planning	SE Integration	Maturity
JADE	Native	Custom	Custom	Custom	Low	High
Jason	Native	BDI	Custom	BDI	Low	Moderate
LangChain	Via agents	Yes	Extensive	Via prompts	Moderate	High
AutoGen	Native	Yes	Moderate	Via dialogue	Low	Moderate
CrewAI	Native	Yes	Moderate	Role-based	Low	Moderate
MetaGPT	Native	Yes	Code-focused	Workflow	Low (SW only)	Moderate
LangGraph	Native	Yes	Via LangChain	Graph-based	Moderate	Emerging

Observations:

- No framework provides native SE tool integration; custom development required
- LLM-based frameworks offer superior reasoning but less multi-agent infrastructure
- Classical platforms offer robust infrastructure but lack modern AI capabilities

- Gap exists for SE-specific multi-agent frameworks

Emerging direction: Hybrid approaches combining classical MAS infrastructure (coordination, lifecycle) with LLM-based reasoning capabilities may address current limitations.

Word count: ~750 words **Subsections:** 4 **Tables:** 1 **References cited:** [71]-[82]

Revision Notes

- [ ] Add more recent frameworks (2024-2025 releases)
- [ ] Expand SE tool integration section with specific APIs
- [ ] Consider adding architecture diagrams for key frameworks
- [ ] Add discussion of framework selection criteria

Section 8: Challenges and Open Problems

Target length: ~1,000 words **Status:** Draft v0.1

---

8. Challenges and Open Problems

Despite promising research directions, substantial challenges impede practical adoption of multi-agent AI systems in systems engineering. This section categorizes and analyzes key challenges.

8.1 Technical Challenges

Scalability. As agent count increases, coordination overhead grows—potentially faster than capability gains. Communication volume, conflict frequency, and convergence time may scale poorly. Research questions: What coordination mechanisms scale effectively? Where are the practical limits of swarm size?

Reliability. LLM-based agents exhibit unpredictable failures—hallucinations, reasoning errors, instruction drift. Multi-agent systems can amplify failures through error propagation or exhibit emergent failure modes. Research questions: How can agent reliability be characterized and bounded? What architectural patterns improve robustness?

Domain knowledge integration. Effective SE support requires deep domain knowledge that current LLMs may lack or represent incorrectly. RAG helps but doesn't solve fundamental knowledge gaps. Research questions: How can domain constraints be reliably encoded? How can physics-based reasoning be integrated with language model capabilities?

Consistency maintenance. Multi-agent systems generating or modifying engineering artifacts must maintain artifact consistency. Concurrent modifications can introduce conflicts; coordination must ensure coherent results. Research questions: What consistency models suit SE artifacts? How should conflicts be detected and resolved?

Performance. LLM inference is computationally expensive; multi-agent systems multiply this cost. Latency may exceed acceptable bounds for interactive applications. Research questions: What efficiency improvements are possible? How should computation be allocated across agents?

8.2 Integration Challenges

Tool integration. Connecting agent systems with SE tools (MBSE platforms, requirements management, PLM) requires substantial integration effort. Proprietary APIs, data formats, and access models complicate integration. Research questions: What integration patterns are effective? Can standardized interfaces emerge?

Process integration. Inserting AI agents into established SE processes raises workflow questions. Where do agents participate? How are agent outputs reviewed? How do agent activities align with milestone gates? Research questions: What process adaptations are needed? How should traditional and AI-augmented processes coexist?

Data integration. Agent systems require access to engineering data—requirements, models, test results—often distributed across systems with different formats and access controls. Research questions: How can engineering data be made agent-accessible? What data quality requirements apply?

Legacy system accommodation. Most SE organizations have substantial investments in existing tools and processes. Agent systems must accommodate legacy constraints rather than requiring greenfield adoption. Research questions: What migration paths are practical? How can value be delivered incrementally?

8.3 Human Factors Challenges

Trust calibration. Engineers must develop appropriate trust in agent capabilities—neither over-reliance nor excessive skepticism. Trust should be task-specific and evidence-based. Research questions: How does trust develop? What factors support appropriate calibration? How should trust be maintained as capabilities evolve?

Oversight effectiveness. Human oversight of agent activities becomes challenging as agent count and activity rate increase. Engineers cannot review every agent output; effective oversight requires attention management. Research questions: What oversight models are effective? How should agent outputs be prioritized for review?

Skill evolution. Working effectively with agent systems requires skills many engineers lack: formulating effective prompts, evaluating AI outputs, diagnosing agent failures. Research questions: What skills are needed? How should training be structured? How do required skills evolve with technology?

Cognitive load. Managing agent systems imposes cognitive demands: understanding agent state, interpreting agent outputs, coordinating agent activities. These demands may offset productivity gains. Research questions: How can cognitive load be managed? What interface designs minimize burden?

Role clarity. As agents assume engineering tasks, human roles must evolve. Unclear role boundaries create confusion about responsibilities. Research questions: How should human-agent roles be defined? How should accountability be allocated?

8.4 Organizational Challenges

Governance frameworks. Organizations need frameworks governing AI involvement in engineering. What decisions can agents make? How are agent outputs validated? Who is accountable for agent contributions? Research questions: What governance models are appropriate? How should governance evolve with capability?

Certification implications. In regulated industries, AI involvement raises certification questions. How do AI contributions affect system certification? What evidence is required for AI tool qualification? Research questions: What certification frameworks apply? How should evidence be collected and presented?

Liability and accountability. When AI-influenced engineering decisions lead to problems, liability questions arise. Current legal frameworks may not adequately address AI contributions. Research questions: How should liability be allocated? What organizational structures support appropriate accountability?

Change management. Adopting agent systems represents significant organizational change. Resistance, skill gaps, and process disruption must be managed. Research questions: What adoption approaches succeed? How should change be paced?

Economic justification. Agent system adoption requires investment; organizations need evidence of return. Current evidence is limited and context-dependent. Research questions: What value propositions are compelling? How should benefits be measured?

8.5 Challenge Prioritization

Table 5 assesses challenges by severity and tractability.

Challenge	Severity	Tractability	Priority
Reliability	High	Moderate	Critical
Domain knowledge	High	Moderate	Critical
Trust calibration	High	Moderate	Critical
Governance	High	Low	High
Scalability	Moderate	Moderate	High
Tool integration	Moderate	High	High
Certification	High	Low	High
Oversight	Moderate	Moderate	Medium
Performance	Moderate	High	Medium
Skills evolution	Moderate	High	Medium

Critical challenges (high severity, moderate tractability) warrant immediate research investment. High-priority challenges are either severe or tractable and should receive sustained attention.

Word count: ~900 words **Subsections:** 5 **Tables:** 1

Revision Notes

- [ ] Add specific examples for each challenge
- [ ] Cite relevant papers addressing each challenge
- [ ] Consider expanding organizational challenges
- [ ] Add cross-references to solutions in research directions

Section 9: Research Directions

Target length: ~800 words **Status:** Draft v0.1

---

9. Research Directions

Based on the survey findings and challenge analysis, this section proposes research directions organized by time horizon.

9.1 Near-Term Research Priorities (1-3 years)

Benchmark development. The field urgently needs standardized benchmarks for evaluating multi-agent AI systems in SE contexts. Near-term efforts should:

- Develop task suites for key SE processes (requirements analysis, architecture evaluation, test generation)
- Establish evaluation protocols enabling cross-study comparison
- Create shared datasets with realistic complexity and domain diversity
- Build community infrastructure for benchmark hosting and evaluation automation

Reliability characterization. Understanding when and how agent systems fail is essential for responsible deployment:

- Develop taxonomies of failure modes for multi-agent SE systems
- Create methods for detecting and diagnosing agent failures
- Establish reliability metrics appropriate for SE applications
- Design architectural patterns that improve robustness

Domain knowledge grounding. Improving agent access to authoritative domain knowledge:

- Develop retrieval approaches specialized for SE knowledge sources (standards, handbooks, domain literature)
- Create methods for encoding domain constraints as verifiable bounds
- Investigate hybrid architectures combining LLM reasoning with physics-based analysis
- Build domain-specific knowledge bases in machine-accessible formats

Tool integration patterns. Enabling practical integration with SE tools:

- Develop integration reference architectures for common SE tools (MBSE, requirements management, PLM)
- Create abstraction layers reducing tool-specific development effort
- Establish protocols for agent-tool interaction
- Build open-source integration examples accelerating adoption

9.2 Medium-Term Research Agenda (3-7 years)

Coordination at scale. Enabling effective coordination among larger agent populations:

- Develop theoretical models predicting coordination overhead as a function of swarm characteristics
- Design hierarchical and hybrid coordination architectures that scale
- Investigate emergent coordination mechanisms with predictable properties
- Create simulation environments for studying large-scale swarm behavior

Human-AI teaming. Understanding and supporting effective collaboration:

- Develop models of human-agent team performance predicting when collaboration helps
- Design interfaces supporting effective oversight of multi-agent activities
- Investigate trust development and calibration in engineering contexts
- Create training approaches preparing engineers for agent collaboration

Evaluation methodology. Advancing how we assess multi-agent SE systems:

- Develop methods for evaluating coordination quality beyond task completion
- Create longitudinal evaluation approaches for lifecycle support
- Establish metrics for human-AI team effectiveness
- Design evaluation frameworks accommodating evolving capabilities

Governance frameworks. Establishing structures for responsible deployment:

- Develop accountability frameworks clarifying human and agent responsibilities
- Create audit and provenance mechanisms for agent contributions
- Investigate certification approaches for AI-assisted engineering
- Design organizational models supporting appropriate AI involvement

9.3 Long-Term Research Vision (7+ years)

Collective engineering intelligence. Moving beyond agent systems as tools toward genuine collaborative intelligence:

- Understand how human-AI teams can achieve capabilities neither could alone
- Investigate emergent properties of sustained human-AI engineering collaboration
- Develop frameworks for knowledge accumulation across projects and organizations
- Create approaches for preserving and transferring engineering expertise through AI systems

Self-improving systems. Enabling agent systems that improve from experience:

- Develop learning mechanisms improving agent performance from deployment experience
- Create approaches for agents to identify and address their own limitations
- Investigate safe self-modification within bounded authority
- Design systems that improve while maintaining reliability

Engineering automation boundaries. Understanding appropriate automation scope:

- Investigate which SE activities benefit from human execution versus AI execution
- Develop principled approaches for allocating tasks between humans and agents
- Create frameworks for evolving automation boundaries as capabilities and trust develop
- Study long-term implications of automation for engineering practice and workforce

9.4 Cross-Disciplinary Opportunities

Progress requires contributions from multiple research communities:

AI and machine learning: Foundation model capabilities, multi-agent coordination, tool use, reliability

Software engineering: Development methodologies, testing, tool integration, process models

Systems engineering: Domain expertise, process knowledge, practitioner insight, adoption requirements

Human factors: Trust, teaming, interface design, cognitive load, skill requirements

Organizational science: Governance, change management, accountability, workforce evolution

Philosophy and ethics: Responsibility allocation, appropriate automation, societal implications

Sustained progress requires cross-disciplinary collaboration—AI researchers understanding SE practice, SE practitioners guiding AI development, human factors researchers studying collaboration, organizational scientists addressing adoption.

9.5 Community Building

Realizing the research agenda requires community infrastructure:

Research consortia. Establishing focused research programs bringing together AI and SE researchers with industrial partners

Shared infrastructure. Building common benchmarks, datasets, and evaluation platforms

Publication venues. Developing venues spanning AI and SE communities

Educational programs. Creating curricula preparing researchers for cross-disciplinary work

Industrial partnerships. Engaging practitioners in research direction setting and validation

---

Word count: ~780 words **Subsections:** 5

Revision Notes

- [ ] Add specific research questions for each direction
- [ ] Cite existing work addressing each area
- [ ] Consider adding funding program alignment
- [ ] Add discussion of research methodology recommendations

Section 10: Conclusion

Target length: ~300 words **Status:** Draft v0.1

---

10. Conclusion

This survey has provided a systematic examination of multi-agent AI systems for systems engineering, synthesizing contributions from artificial intelligence, software engineering, systems engineering, and human factors research communities.

Summary of Findings

We traced the foundations of this emerging field through multi-agent systems, swarm intelligence, and LLM-based agentic AI, establishing the rationale for multi-agent approaches to systems engineering support. Our taxonomy of agent architectures identified the shift from classical BDI and reactive designs toward LLM-based agents with tool use and memory capabilities. Analysis of coordination mechanisms revealed trade-offs between predictability and flexibility, scalability and control, that must be navigated for SE applications.

Mapping applications to ISO 15288 technical processes revealed concentrated maturity in requirements analysis and test generation, with substantial gaps in other lifecycle phases. Evaluation methods and benchmarks remain underdeveloped, limiting rigorous progress assessment. Tools and frameworks are maturing rapidly, though SE-specific integration remains limited.

We identified critical challenges in reliability, domain knowledge integration, trust calibration, and governance—challenges that must be addressed for responsible adoption. Our research agenda prioritized benchmark development, reliability characterization, and human-AI teaming for near-term attention, with coordination at scale and governance frameworks as medium-term priorities.

Key Takeaways

For researchers, this survey provides organized access to a fragmented literature and identifies high-priority gaps warranting investigation. For practitioners, it offers a map of current capabilities and realistic assessment of maturity levels. For the broader community, it establishes common vocabulary and frameworks enabling cumulative progress.

Looking Forward

The field stands at an inflection point. Foundational capabilities exist; challenges are identified; directions are clear. Progress requires sustained research investment, cross-disciplinary collaboration, and engagement between researchers and practitioners. The potential prize—substantially augmented engineering capability for addressing society's complex challenges—justifies the effort required.

We invite the community to build upon this survey, address the gaps identified, and realize the potential of multi-agent AI for systems engineering.

Word count: ~330 words

Revision Notes

- [ ] Ensure conclusion reflects survey content accurately
- [ ] Consider adding specific quantitative summary
- [ ] May adjust emphasis based on final survey findings