Agents Unleashed: Exploring the Terminology, Design, Power of GenAI Agents, and Agentic Systems

@JohnAlexander,

Content Lead, Azure AI App Development, Microsoft

<DevSum>

active SOLUTION Agria Djurförsäkring VONAGE Part of Ericsson RaySearch Laboratories Duende. bulbul SOFTRONIC Polisen

@JohnAlexander

- Content Lead, Azure AI App Development, Microsoft

https://aka.ms/azai

- Tutor, University of Oxford
- Ex-Microsoft, Autonomous Systems Advocacy Team
- 6 years of AI/ML
- 25 years of Software Development, Architecture,
- 3 years of Generative AI

There is a resource sheet for all links used in this session (and more for further exploration) at https://github.com/JRAlexander/agents-unleashed/

Learning focus today

- Level-set on terms
- Explore key design tips
- Explore an Agentic product development process example

Part I: Level Set

What’s an Agent?

- “AI-powered agents are an emerging field with no **established** theoretical frameworks for defining, developing, and evaluating them.” - Chip Huyen’s Blog, Agents

What’s an agent (Traditional definition)?

- “An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**.” - Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig,1995, Prentice-Hall

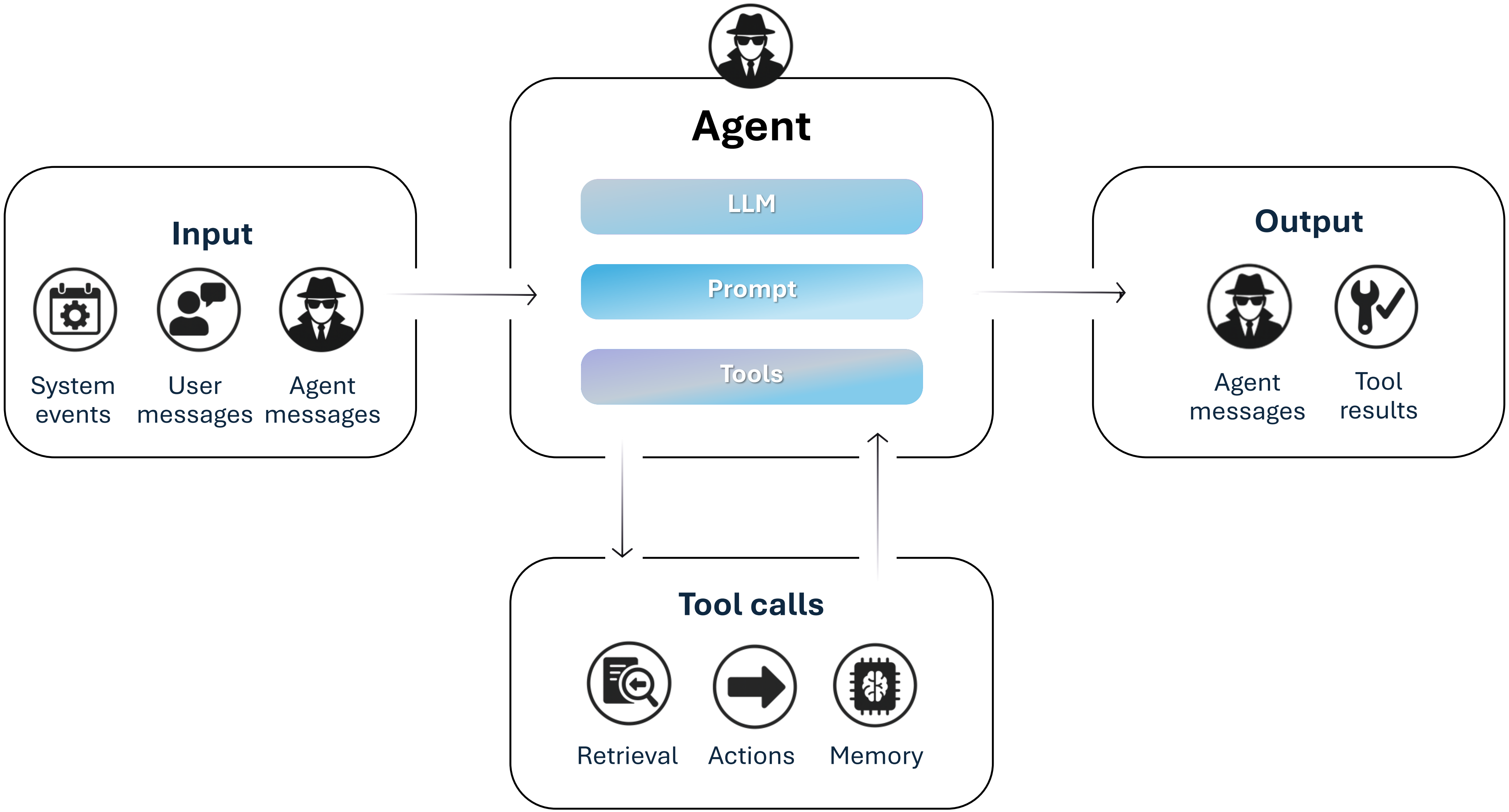
- Two key pieces:
  - The **environment** it operates in (defined by its use case)
  - The **actions** it can perform (defined by its environment)

- Tools and Planning

Terms

Agent	A software entity that performs tasks autonomously, often using AI to make decisions and interact with the environment.
Workflow	A system where LLMs and tools are orchestrated through predefined code paths.
Metaprompting	A technique using an LLM to generate or improve prompts
Model Context Protocol	An open protocol for connecting AI apps to tools, APIs, and data sources.
Agent 2 Agent Protocol	An open standard for AI agent communication and collaboration across different platforms and frameworks, regardless of their underlying technologies.
Deterministic system	A system that always produces the same output for a given input.
Probabilistic system	A system that produces a probability distribution over possible outcomes and will likely have different outcomes given the input.
Multi-Agent	A system of multiple autonomous agents that interact or work together to achieve individual or shared goals.

Environment



Agent

LLM

Prompt

Tools

Input

System events

User messages

Agent messages

Output

Agent messages

Tool results

Tool calls

Retrieval

Actions

Memory

Tool calling – Function calling

Function calling allows large language models (LLMs) to **invoke external functions or APIs** based on natural language input.

Why It Matters

- Function calling transforms LLMs from **static text generators** into **dynamic, tool-using agents**.

Tool calling – Model Context Protocol (MCP)

An open protocol for connecting AI apps (clients) to tools, APIs, and data sources (servers).

Why It Matters:

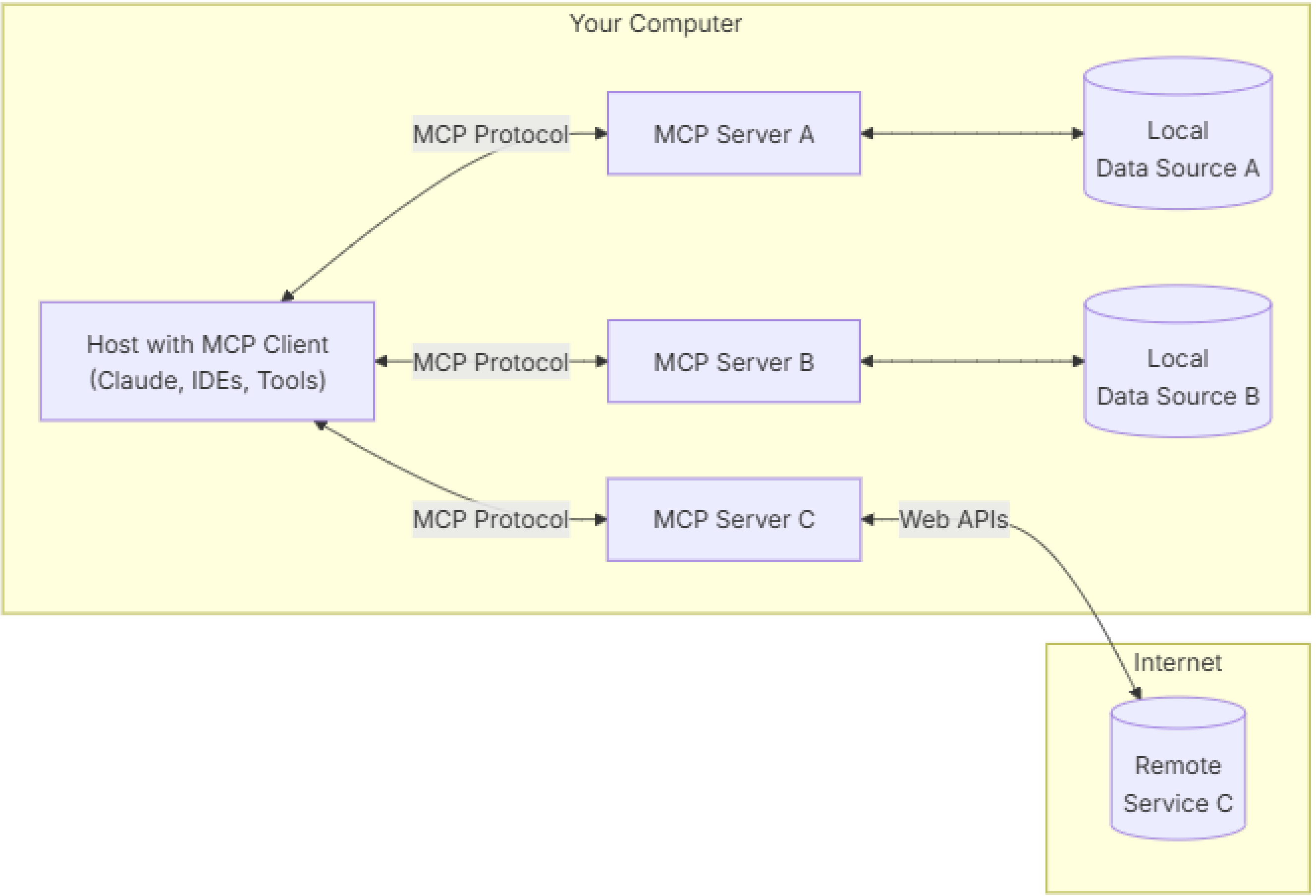
- Standardizes tool/data access
- Creates a single interface all apps and tools can use
- Inspired by APIs and LSP (Language Server Protocol)

MCP Architecture

- MCP Host: The environment where the agent logic lives (your own app, Azure OpenAI, Copilot Studio)
  - Agent: The LLM-based reasoning unit (e.g., Azure OpenAI GPT-4 model)
  - MCP Client: Communication layer that speaks the Model Context Protocol
- MCP Server: Provides access to tools, APIs, and data (e.g., Azure CLI, resource graphs)
- Data Sources:
  - Local
  - Remote

Flow: User → Agent → MCP client ↔ MCP Server → Services

MCP Architecture



MCP Interfaces

Three Interfaces (provided by server, consumed by client):

- **Tools** – model-controlled - Functions that can be called by the LLM (with user approval)
- **Resources** – application-controlled - File-like data read by clients (like API responses or file contents)
- **Prompts** – user-controlled - Pre-written templates that help users accomplish specific tasks

Tool calling – MCP vs Function call

Use MCP when:
- You want reusable tool endpoints across agents
- You need **fine-grained control**, **auditing**, or **identity-aware access**
- You are building for **enterprise or secure Azure scenarios**

Use function calling when:
- You are prototyping or need **tight integration** within a single model call
- Your tools are **lightweight** and don’t need centralized governance

Multi-Agents

- A system of multiple autonomous agents that interact or work together to achieve individual or shared goals.

- Multi-agent systems are used in fields like artificial intelligence, distributed computing, economics, and robotics to model and solve complex problems.

Key Characteristics of Multi-Agent Systems

Autonomy:

- Each agent operates without direct human intervention, controlling its own actions based on its goals.

Decentralization:

- No single agent has complete control over the entire system.
- Agents work independently or collaboratively, contributing to the overall system behavior.

Interaction:

- Agents communicate and cooperate, compete, or negotiate with each other to achieve their goals.

Distributed Problem Solving:

- The system can solve problems that are too complex for an individual agent by dividing tasks among multiple agents.

Adaptation:

- Agents can learn and adapt their behavior over time based on experiences or environmental changes.

Agent 2 Agent (A2A) protocol

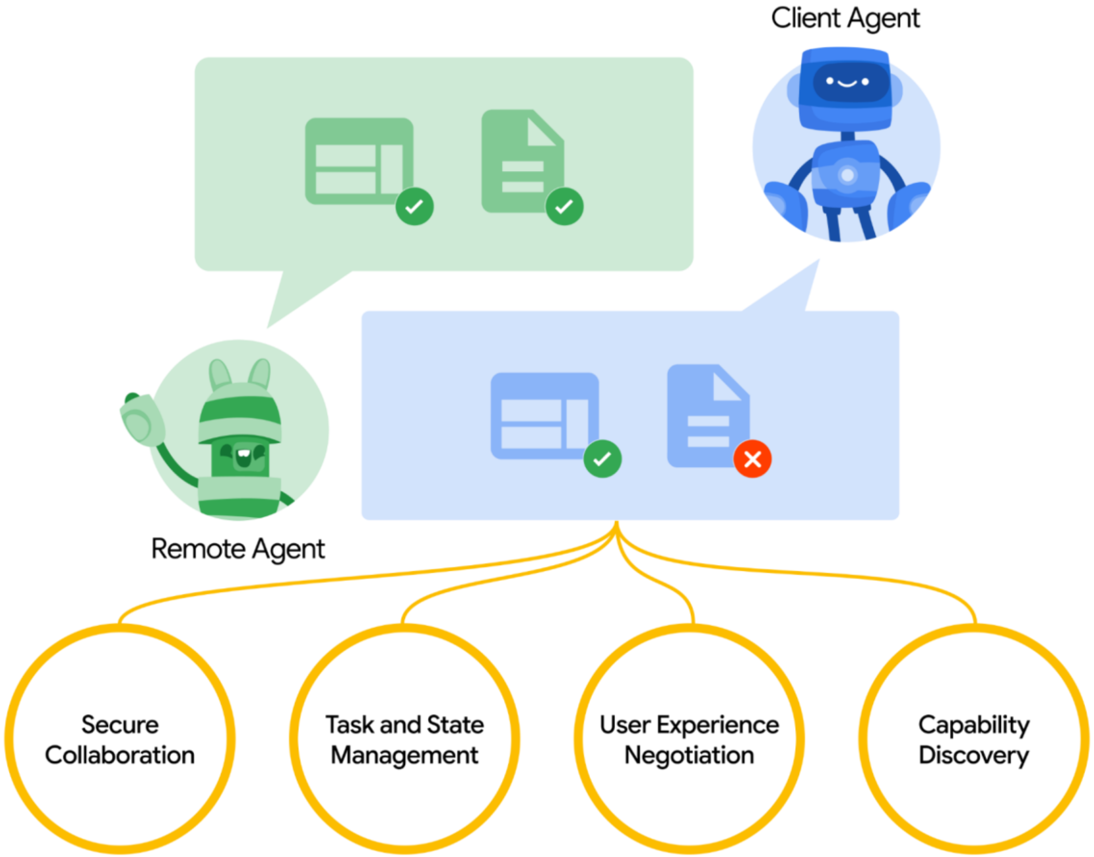
A2A facilitates communication between a 'client' agent and a 'remote' agent through a structured process

Capability Discovery
Agents advertise their capabilities using an 'Agent Card' in JSON format, enabling other agents to identify the best agent for a task.

Task Management
Communication is oriented towards task completion, with a defined lifecycle that can be completed immediately or over time.

Collaboration
Agents can send messages to communicate context, replies, artifacts, or user instructions.

User Experience
Messages include 'parts' with specified content types, allowing agents to negotiate the correct format and UI capabilities.



What essential skills should agent developers master?

- **Workflow Breakdown**: Clearly defining business processes into manageable tasks.
- **Data Integration & Plumbing**: Using frameworks and tools like MCP for smooth data integration.
- **Evaluation Frameworks**: Quickly setting up systematic evals to trace and manage improvements efficiently.
- **Rapid Decision-Making**: Developing instincts to identify which agent components to refine or abandon quickly.

Source: Andrew Ng: State of AI Agents | LangChain Interrupt, May 2025

Part II -   Key Design Principles

Don’t build agents for everything

Use agents only when the task *requires* exploration and autonomy.

- Task complexity justifies autonomy.
- Task value offsets high token cost.
- Agent can perform critical capabilities reliably.
- Errors are tolerable or detectable.

How do I know if my task is worth building an agent for?

Use this checklist:

- Is the task ambiguous?
- Does it justify high token cost?
- Can the agent act effectively?
- Can I detect/fix errors quickly?

Designing AI Agents with the Jobs To Be Done Framework

JTBD Core Principle

"People don’t buy products; they hire them to get a job done.“

Applied to AI:

"Users don’t engage with agents for AI’s sake; they want specific jobs completed."

Designing AI Agents with the Jobs To Be Done Framework

JTBD Step	AI Agent Design Equivalent
Define the core job	What's the agent hired to do?
Functional/emotional/social dimensions	What user outcomes must be met?
Desired outcomes	What defines agent success?
Struggling moments	Where do users get stuck today?
Capabilities design	Build features to resolve struggles
Hire/fire criteria	Why users continue or stop using the agent?

JTBD Checklist for AI Agents

✅ Have we defined the job the user is hiring the agent for?

✅ Do we understand emotional and functional needs?

✅ Are agent outputs tied to real progress?

✅ Does the agent reduce or eliminate user struggles?

✅ Can we measure success as job completion?

Choosing the right AI model for your task

The best model depends on your use case:

- For balance between cost and performance,
  try GPT-4.1 or Claude 3.7 Sonnet.
- For fast, low-cost support for basic tasks:
  try o4-mini or Claude 3.5 Sonnet.
- For deep reasoning or complex coding challenges
  try o3, GPT-4.5, or Claude 3.7 Sonnet.
- For multimodal inputs and real-time performance:
  try Gemini 2.0 Flash or GPT-4.1.

Iterate reality!

- Sit with real users (e.g., in logistics or support centers).
- Learn their actual workflows / processes.
- Encode their logic into prompts and evals.
- Iterate quickly with them in the loop. - FEEDBACK

Start with the three core agent components

- Environment — What the agent acts upon.
- Tools — Interfaces for actions + feedback.
- Prompt — Goals, constraints, behavioral guardrails.

Longer prompts (even 6+ pages) are okay if structured cleanly:

- Role definition
- Task plan
- Tool usage
- Output format
- Reasoning scaffolding
- Worked examples

Treat Prompts Like Code (Prompt as Spec)

- Structure prompts using roles, tasks, constraints, and formats—just like defining a software interface.
- Use clear markdown or XML-like formatting for parsing and stability.
- Prompt should act as an API contract between your agent and your system.

Test Prompts and Tools with the Model Itself

Ask the model:

- Is this prompt clear?
- Is the tool easy to use?
- Are parameters missing?

Think Like the Agent

Constrain yourself to its **context window**:

- Same token limits
- No memory beyond current state
- No visual continuity

Practical exercise:

Try completing a task with only a screenshot + brief description.
This will reveal missing context in your agent's design.

Evaluation & Testing

- Start simple, with clear success metrics.
- Iterate and validate agent effectiveness early and often.
    - Run evals across tool versions
    - Test with fallback scenarios
- Compare agent output pre/post update

MCP Server – Getting Started

- Start with tools, then add prompts/resources
- Separate logic: model vs app vs user
- Favor dynamic interpolation
- Push business logic to server

MCP Server - Treat Tools Like Prompts

- Write **descriptive tool documentation**.
- Remember: Tool specs are part of the prompt.
- Poor tool design = poor agent performance.

MCP Auth & Security

- Use OAuth 2.0 (natively supported)
- Let server manage access tokens
- Trust but verify: vet external servers

MCP Debugging & Observability

- Use **Inspector** for logs and tool tracing
- Return metadata from tools
- Document capabilities clearly

Just like an API, a good MCP server includes clear tool names, capabilities, and optional annotations.

Mitigate Risk in Early Deployments

- Use read-only or human-in-the-loop phases.
- Scope tasks narrowly at first.

AI Agent Development Best Practices

- ✅ **Understand the Core Difference: Agent vs. Workflow**
  Know when to use autonomous, iterative agents vs. fixed, linear workflows.
- ✅ **Think Like the Model**
  Simulate the model’s limited context and reasoning ability when designing prompts and environments.
- ✅ **Treat Tools Like Prompts**
  Provide clear, documented, and well-named tools—these are part of the prompt context too.
- ✅ **Measure Everything**
  Build with evaluation in mind—test whether your agent actually improves outcomes.
- ✅ **Pick the Right Problems**
  Use agents where tasks are complex and valuable, but the cost of error is low (e.g., search, coding).
- ✅ **Design for the Future**
  Structure your systems so they improve as models improve, not break under smarter behavior.

Agent Insights Summary

Insight	Takeaway
Agents ≠ Workflows	Use agents for exploration, workflows for control.
Scope = Trust	Start with safe use cases; expand once validated.
Simplicity Wins	Start with a minimal architecture and optimize later.
Tool UX Matters	Agents benefit from intuitive, constrained tools.
Verifiability is Key	Build agents in domains where you can test outcomes.
Context is Everything	Understand agent failures by reproducing their context view.
Use the Model to Debug Itself	Agents can audit prompts and decisions when prompted.
Does it work in Multi-Agent use cases?	Plan for asynchronous agent-to-agent protocols.
Meta-Tools Enable Scale	Let agents help evolve their own infrastructure.

Key Design Principles for Multi-Agent vs Single-Agent Systems

- Fundamental differences in designing single-agent and multi-agent systems.
- Multi-agent systems require a shift from individual optimization to managing interactions.
- Key principles include communication, coordination, and distributed decision-making
- Addressing these principles leads to efficient, reliable systems capable of complex tasks through collective intelligence.

Communication and Interaction

Single-Agent Systems

- Interacts primarily with the environment.
- No need for inter-agent communication.

Multi-Agent Systems

- Agents must communicate with each other.
- Share information, negotiate, and coordinate actions.

Design Principle

- Establish robust communication protocols
- Ensure reliability, scalability, and security in communication

Coordination and Cooperation

Single-Agent Systems

- Focus on internal coordination for optimizing actions.

Multi-Agent Systems

- Agents need to coordinate actions to achieve shared goals or avoid conflicts.

Design Principle

- Implement coordination mechanisms (centralized planning, distributed consensus).
- Define clear protocols for cooperation and conflict resolution.

Distributed Decision-Making

Single-Agent Systems

- Centralized decision-making within the agent.

Multi-Agent Systems

- Decentralized decision-making; each agent is autonomous.

Design Principle

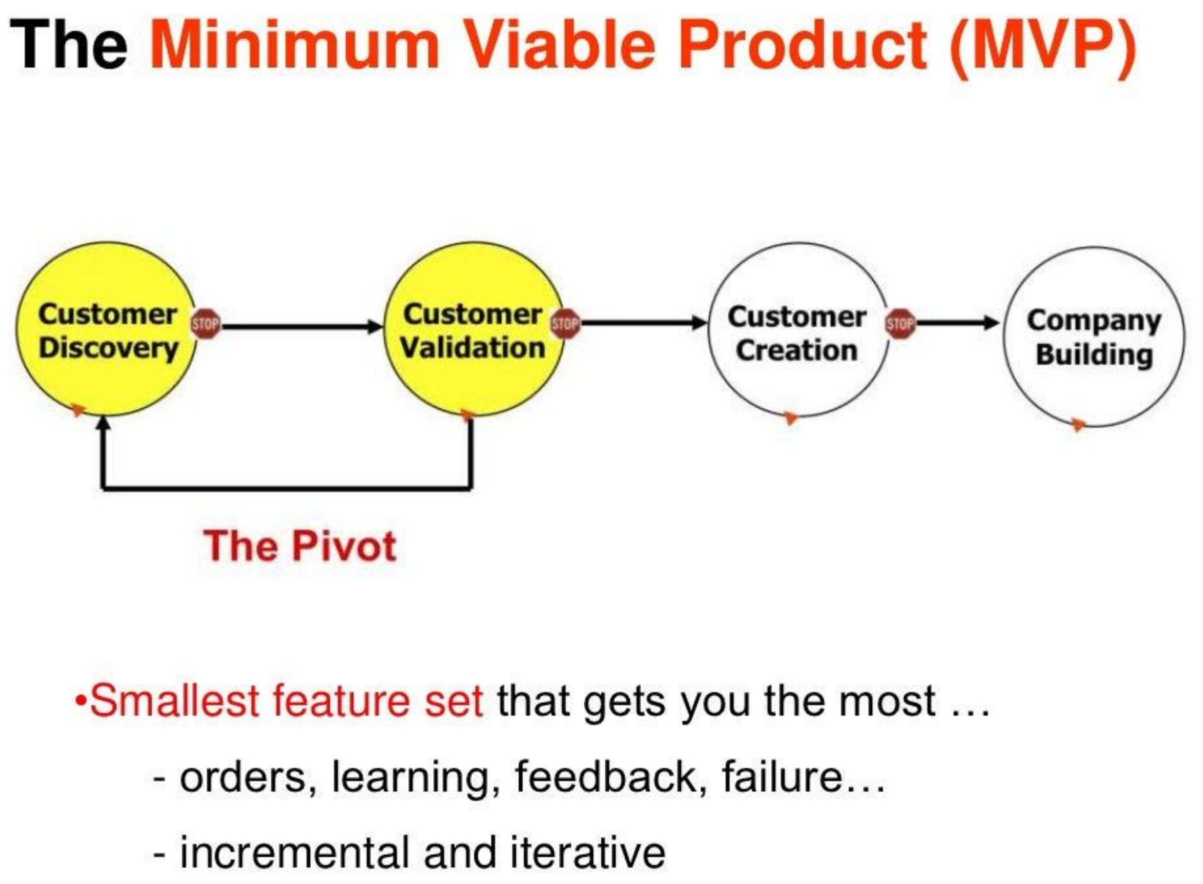
- Design agents to make decisions based on local information.
- Use distributed algorithms for consistency and global objectives

Part III - The Agentic Product Development Lifecycle

Agentic AI Product Development Lifecycle

Section	Purpose
Envisioning	Define the AI agent or MAS high-level vision and business goals strategic alignment.
Ideation	Brainstorm the agent’s features, capabilities, and user scenarios.
Identification of MVP	Define the Minimum Viable Product (MVP) that addresses the core job with minimal resources.
Designing	Develop artifacts and workflows that define the agent’s behavior, goals, and constraints.
Prompting	Develop the initial interaction strategies and prompts for the AI agent.
Data Engineering	Ensure the AI agent has reliable access to necessary data, with attention to context, quality, and scalability.
Customer Development	Validate the AI agent’s assumptions through user feedback and pilot testing.
Building an MVP	Develop and deploy the MVP version of the AI agent.
Measuring Success	Evaluate the agent’s performance and value delivery.
Iteration Loop	Continuously refine and expand the agent’s capabilities based on feedback and performance.

Customer Development and Agents



The Minimum Viable Product (MVP)

- Smallest feature set that gets you the most …
  - orders, learning, feedback, failure…
  - incremental and iterative

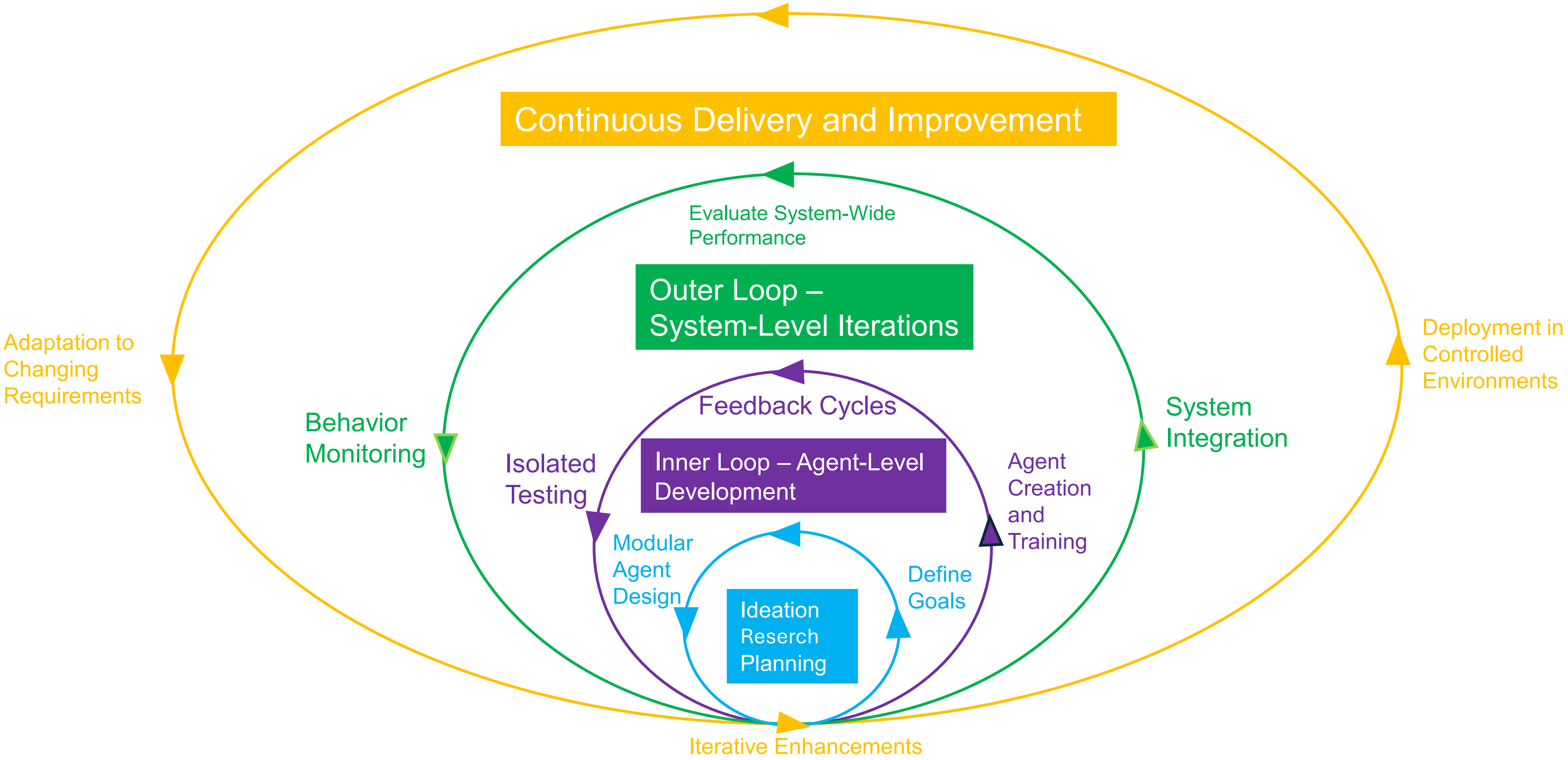
How an Agentic MVP Differs from a Traditional MVP

- Dynamic Adaptation:
  - Agentic MVPs include mechanisms for real-time learning and adaptation
- Data Pipeline Focus:
  - Emphasizes reliable access to structured and unstructured data for decision-making and functionality.
- Feedback Integration:
  - Incorporates continuous feedback loops for refinement during use
- Ethical Design:
  - Explicit consideration for fairness, transparency, and user trust as part of the core design.

Apps and App ecosystems

- Agents aren’t features.
  - They solve or handle a specific task or job.
- When adding agentic aspects to an existing product, think integration with a new app or system.
- A multi-agent system is really an App ecosystem.

Agentic Development journey



Continuous Delivery and Improvement

Outer Loop –
System-Level Iterations

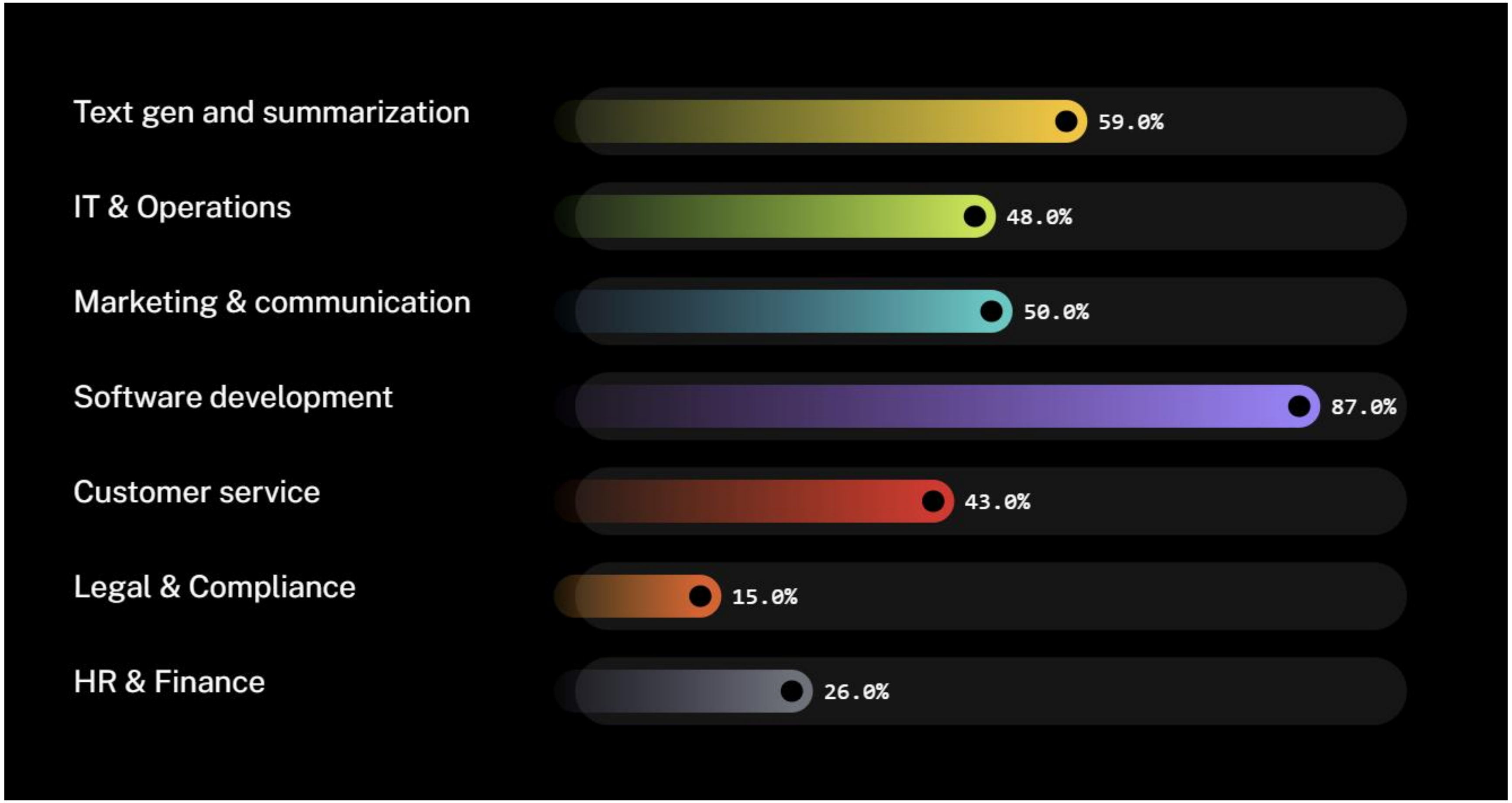
Inner Loop – Agent-Level
Development

Ideation
Reserch
Planning

How an Agentic MVP Differs from a Traditional MVP

- Inter-Agent Workflows (MAS Context):
  - For multi-agent systems, includes basic inter-agent communication and task delegation.
- Generative Features:
  - Incorporates creative outputs or decision-making tailored to user-specific needs.
- Advanced KPIs:
  - Measures performance using adaptive metrics (response accuracy, relevance) rather than only functional or usage-based metrics.

Top use cases for LLMs
Langbase State of AI Agents



Source: https://langbase.com/state-of-ai-agents#usecases

Questions?

All materials at https://github.com/JRAlexander/agents-unleashed/

Thanks!

@johnalexander

Resources

- Andrew Ng: State of AI Agents | LangChain Interrupt
- Tips for building AI agents
- Building Agents with Model Context Protocol - Full Workshop with Mahesh Murag of Anthropic
- AI prompt engineering: A deep dive
- State-Of-The-Art Prompting For AI Agents
- Choosing the right AI model for your task - GitHub Docs