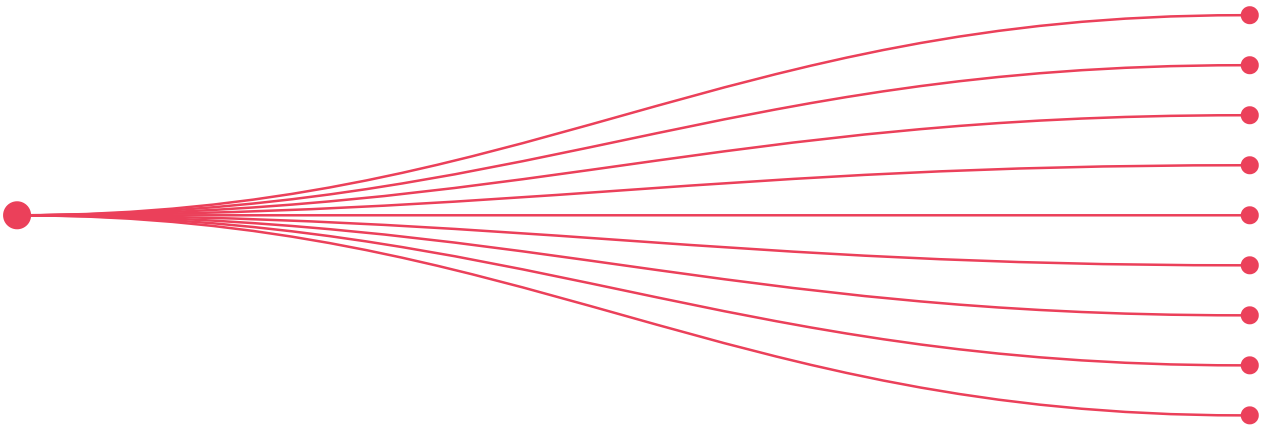


Rethinking Software Engineering for the Agentic Era

Why AI agents demand more than new interfaces, and how IDEA can connect user intentions to business outcomes.

Panos Kontopoulos
Chief Executive Officer · MOCTA Commerce Suite

Executive Summary

What if we stopped designing every possible user journey and instead engineered the capabilities to fulfill many different intentions? That question is the starting point for this essay.

Conventional software relies on predefined screens and workflows. This approach is essential for control and reliability, but it cannot economically anticipate every combination of user objectives, context, and exceptions. AI agents introduce an alternative interaction layer: users express desired outcomes, while the application retrieves context, selects permitted business capabilities, and composes a suitable journey.

Jobs to Be Done (JTBD) offers a disciplined starting point by identifying the progress people seek. This essay proposes Intent-Driven Engineering of Applications (IDEA) as a way to connect those jobs to contextual intents, reusable capabilities, governed agent tools, dynamic journeys, and verified outcomes. Examples from hospitality, supermarket shopping, and education demonstrate the underlying model.

The central proposition is not software without structure. It is software that preserves trusted rules, data, and transactions while giving people more flexibility in how they achieve meaningful business outcomes.



ABOUT THE AUTHOR

Panos Kontopoulos

Chief Executive Officer · MOCTA Commerce Suite

Panos works at the intersection of digital business strategy, product development, customer experience, and enterprise commerce. As Chief Executive Officer of MOCTA Commerce Suite, he leads its growth strategy, including product positioning, ecosystem development, and international market expansion. His interests include AI-enabled experiences and the next generation of digital business applications.

p.kontopoulos@moctasuite.com linkedin.com/in/pkontopoulos

01

The Software Paradigm We Have Been Living With

I keep coming back to one question: what if we stopped treating the user journey as the primary thing we design?

We have become very good at telling people how to move through software. AI agents now give us an opportunity to rethink the relationship: people could express the progress they need to make, while applications assemble the right path through trusted business capabilities. The challenge is not only technical. It is also about how we discover which outcomes deserve to be supported.

For decades, we have built business applications by identifying the data an organization needs, defining its business rules, designing workflows, and placing screens on top. Analysts describe the requirements. Designers arrange the interactions. Engineers implement the logic. Users learn to navigate the result.

This approach has taken us a long way. A well-designed ERP, CRM, commerce platform, or booking system makes complex operations repeatable and auditable. It gives people consistent controls, enforces permissions, and prevents invalid transactions. For stable, high-volume activities, a predefined flow is often exactly what we need.

But there is an assumption at the center of this model: the people designing the software can anticipate the journeys that users will need to follow. We do not simply model what the business can do. We prescribe how people should move through those capabilities to get something done.

02

The Problem With Designing Every Possible Journey

Real business situations are rarely as tidy as our diagrams. A merchant trying to rescue a delayed order may face a stock shortage, an unavailable courier, a customer deadline, or a refund restriction. Each condition changes what an appropriate response looks like. Multiply the goals, states, constraints, exceptions, and permissions, and the number of legitimate paths grows quickly.

That creates what I think of as a **completeness problem**. Even excellent software teams cannot economically anticipate every useful combination. What gets built reflects the current requirements, the experience of the analysts and engineers involved, and the budget available at that moment. The result is not necessarily poor software; it is software with boundaries drawn by what we knew when we designed it.

Extending those boundaries can be expensive. A new business situation may call for another page, another workflow, another integration, and another release. Meanwhile, users compensate by exporting spreadsheets, sending emails, asking colleagues, or moving between applications. They know the outcome they want, but they must translate that outcome into the language and navigation of the system.

I am not arguing that fixed journeys should disappear. They are ideal when processes are stable, regulated, or already optimized. My question is whether we should continue treating the predefined journey as the only way users can access business capabilities.

The result is not necessarily poor software; it is software with boundaries drawn by what we knew when we designed it.

03

AI Changes the Equation: From Predefined to Composed Journeys

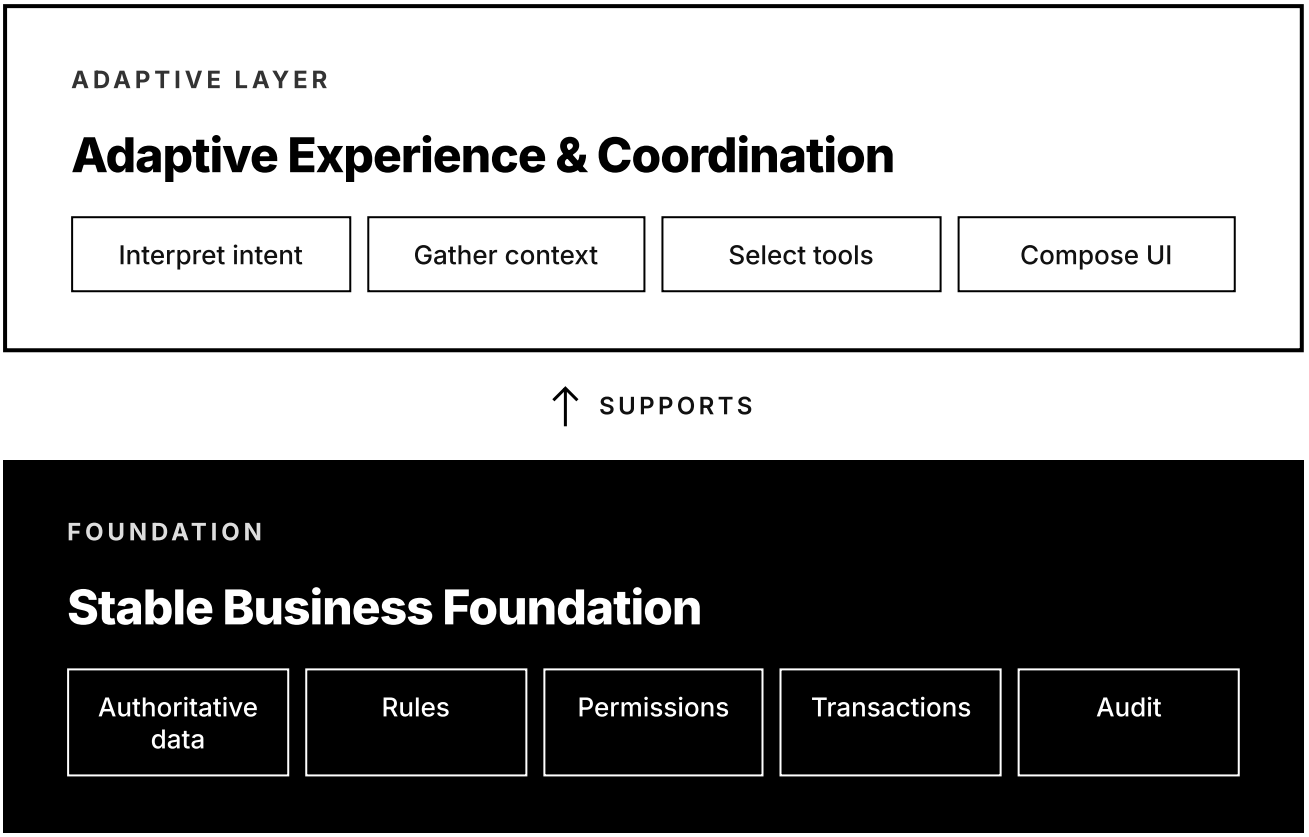
AI agents introduce a different possibility. Instead of requiring the user to discover the correct sequence of screens, an application can begin with an objective: **“Help me resolve the orders that will arrive late today.”** The system can interpret the request, retrieve relevant context, select suitable capabilities, evaluate alternatives, and prepare an appropriate response.

One case may call for a shipment-status investigation. Another may need inventory checks and a replacement option. A third may require a human decision because the requested compensation exceeds a policy limit. The user has expressed the job; the journey is composed according to the situation.

This is more than putting a chatbot on top of existing software. It is a change in the application model: a stable, trusted business foundation combined with a more adaptive layer for coordination and interaction. Agents may guide users through existing screens, assemble contextual workspaces, or present generative UI elements such as comparisons, timelines, forms, and approval panels.

The user has expressed the job; the journey is composed according to the situation.

Figure 1. A stable business foundation supports an adaptive experience and coordination layer.



The foundation must remain solid. Source systems should still own authoritative data. Domain services should enforce permissions, calculations, and valid state transitions. Sensitive actions should require appropriate authorization and, where necessary, explicit human approval. **An agent may decide which permitted tool to call; it should not get to rewrite the rules of the business.**

That is also why I do not see generative UI as an end in itself. Sometimes the best experience is a familiar form. Sometimes it is a dynamic interface. The meaningful advance is the freedom to use the most appropriate interaction for the job.

04

Jobs to Be Done: Start From What People Want to Achieve

This brings me to a methodology that I believe deserves a much more central role in software engineering: Jobs to Be Done (JTBD).

JTBD is a framework for understanding the progress people are trying to make in a particular situation and the outcome they want to achieve. A job is not a feature or a screen. It is the underlying reason a person needs help.

A salesperson does not fundamentally want to filter opportunities; they want to focus their effort where it can make the greatest commercial difference. A resort guest does not want to navigate separate booking widgets; they want to enjoy a well-organized stay. A student does not want to browse a course catalogue for its own sake; they want to make reliable progress toward graduation.

A useful job story captures the situation, the desired progress, and the outcome. From that relatively stable job, many contextual intents may emerge.

THE JOB STORY

When something happens, **I want to** make specific progress, **so I can** achieve something meaningful.

JTBD is not an AI architecture and it does not replace domain modeling, UX research, or requirements engineering. It provides something prior to all of them: a disciplined way to decide which progress the system should help its users make.

05

From Jobs to Business Outcomes: An Engineering Method

I have been exploring a practical synthesis I call **Intent-Driven Engineering of Applications (IDEA)**. It is a proposed methodology, not a formal industry standard. The idea is to create a traceable path from the user's job to an outcome that the application can actually deliver.

The chain begins with a Job to Be Done, moves to the intent expressed in a particular moment, then breaks that intent into goals and constraints. From there, we identify the business capabilities needed: the ability to retrieve customer context, assess a risk, check availability, create a task, or execute a permitted transaction. We expose those capabilities through bounded, governed agent tools. The agent can then assemble a contextual journey, and the system can verify the business outcome.

Take a CRM example. The durable job is to maintain momentum in customer relationships. The immediate intent may be, "Help me organize my customer follow-ups this week." The agent needs to understand the account context, review recent activity, find neglected relationships, and propose sensible next steps. It might use tools such as `get_account_context`, `get_tasks`, and `create_task`. The user reviews the proposed actions before they are committed.

The important architectural move is not to create one agent for every journey. It is to build a **reusable capability layer**. Customer context can support follow-up planning, account health reviews, meeting preparation, and growth opportunities. The same trusted tool can participate in many different journeys.

The same trusted tool can participate in many different journeys.

Equally important, the methodology must start with evidence. Interviews and observation tell us which jobs matter. Domain analysis reveals the rules and constraints. Engineering determines which capabilities are available, which must be added, and which actions can safely be automated. Finally, evaluations measure whether the intended progress actually happened.

Figure 2. From Jobs to Verified Outcomes

The seven-layer model bridges human needs and governed technical execution. Each layer makes a different kind of decision explicit, and the same capabilities can support many intents.

#	LAYER	CRM EXAMPLE
1	**Job to Be Done**	Maintain relationship momentum
2	**User Intent**	“Help me organize customer follow-ups”
3	**Goal Decomposition**	Understand context, assess activity, plan actions
4	**Business Capabilities**	Customer context, activity analysis, task management
5	**Agent Tools**	`get_account_context` / `get_tasks` / `create_task`
6	**Dynamic Journey**	Propose, review, approve, execute
7	**Business Outcome**	A verified, actionable follow-up plan

The agent selects and composes a suitable path. The business foundation validates permissions, state, calculations, and transactions. **The measure of success is the outcome achieved, not the number of screens or tool calls.**

06

Three Industries, Three Jobs, Many Possible Journeys

The appeal of this model is that its structure is reusable across industries even when the business logic is completely different. Consider three examples.

Hospitality	**Make the Most of My Stay** A guest wants to plan a family day at a resort, not coordinate dining, children's activities, and spa appointments across separate systems. An agent could combine stay context, age-appropriate options, live availability, and weather. If it rains or a service is sold out, it can propose an alternative itinerary and ask the guest to confirm the reservations.
Shopping	**Build a Complete Weekly Basket** A household shopper wants to turn planned meals into a suitable basket within a budget. The system can translate meals into ingredients, check verified product facts and allergens, find available substitutes, calculate the total, and offer an appropriate pickup slot. A missing product should change the proposed path, not force the shopper to start over.
Education	**Make Progress Toward Graduation** A student wants to choose courses that keep graduation on track. An agent can combine degree requirements, prior credits, prerequisites, course availability, and timetable conflicts to compare feasible study plans. It may prepare an enrollment choice, but the university's rules and official registration system remain in control.

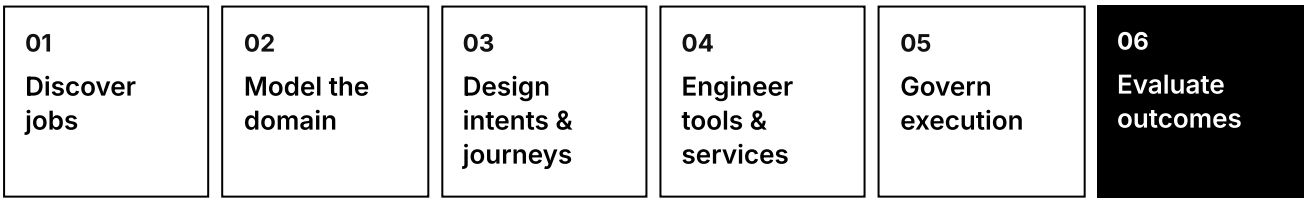
07

A New Opportunity for Digital Services Companies

I also believe this shift could reshape the business of digital services. For many years, engagements have been organized around requirements, screens, modules, integrations, testing, and maintenance. Those disciplines remain essential, but they are no longer the full value proposition.

A more valuable starting point is to help a client discover the jobs their customers and employees need to accomplish, identify which of those jobs are poorly served, and model the reusable capabilities that could support multiple outcomes. Then we can design the tool contracts, permissions, human checkpoints, orchestration patterns, and interfaces that make those capabilities available safely.

That suggests a different engagement model: discover jobs; model the domain; design intents and journeys; engineer tools and services; govern execution; and continuously evaluate outcomes. Instead of delivering a collection of disconnected AI demonstrations, we leave the organization with a capability foundation that becomes more useful as new intents emerge.



It also calls for a different mix of expertise. Business consultants, UX researchers, domain architects, product teams, and engineers need to work much more closely together. The value shifts from merely translating requirements into screens toward helping clients understand, prioritize, and operationalize the progress they want their business to make.

08

The Future Is Not Software Without Structure. It Is Software With More Freedom.

There is a temptation to describe agentic software as if the application will simply understand anything and do everything. I do not believe that is a useful design principle. Reliability still matters. Authorization still matters. Data quality, explicit contracts, and transaction integrity matter even more when software can initiate actions on a user's behalf.

The opportunity is more precise: separate the stable abilities of a business from the particular journeys through which people use them. Keep deterministic workflows where they work best. Introduce adaptive orchestration where context, variation, and cross-system coordination justify it. Let familiar interfaces and generative UI coexist. Give users visibility and control over consequential decisions.

JTBD can help us avoid the trap of building agents just because we can. It asks us to start with the real job, understand its importance, and measure whether the outcome improved. Agents and tools are then a means to an end, not the point of the exercise.

The question I keep coming back to is this:

What if our job was no longer to design every possible journey, but to engineer the capabilities that could fulfill thousands of different user intentions?

01	02	03	04	05
Start with the job.	**Express the intent.**	**Reuse the capability.**	**Govern the action.**	**Verify the outcome.**