buzzy

Is The SaaSpocalypse Real?

A field test of AI-built software in a category where the spreadsheet has teeth

AI can generate an application quickly. The harder question is what the organisation is left to own.

Core thesis: AI-assisted delivery should reduce time to a governed, inspectable application definition - not simply increase the volume of source code a team must later own.

A Field Test Of AI-Built Software In A Category Where The Spreadsheet Has Teeth

The Short Version

The software industry has spent the last year asking whether AI will replace SaaS. That prediction has acquired a memorable nickname - SaaSpocalypse - and the usual amount of certainty that accompanies a good internet noun.

We wanted to test the harder version of the claim. Could AI help build a serious enterprise application, not merely a handsome dashboard with a fictional progress bar? And could it do so without leaving the buyer with a very large pile of generated source code, plus the equally large job of understanding, securing, auditing, and maintaining it?

We chose sales incentive planning as the test case. It is a poor place to bluff. These systems combine sales, CRM, ERP, and HRIS data with effective-dated plans, commission calculations, approvals, reconciliation, disputes, and payroll-adjacent controls. When the payout is wrong, no one is reassured by the elegance of the CSS.

Using Buzzy Builder MCP through Codex, we built a working Sales Incentives & Planning prototype in a few days. It had role-specific experiences, simulated enterprise data sources, live data-backed charts, seller statements, and a calculation waterfall. Most of the application was represented as Buzzy-managed requirements, flows, data model, screen, permission, and workflow artifacts. The bespoke code was concentrated in the places where bespoke code was useful: a deterministic mock integration service and a small set of visualisation widgets.

This is a Buzzy technical case study, not an independent benchmark and not a claim that a mature SPM suite can be replaced in a week. The meaningful result is narrower: a governed application platform can turn a difficult software concept into an inspectable system quickly, without automatically turning every requirement into code that the customer must own forever.

The comparisons in this paper count the customer-specific code required to deliver this scope. They do not pretend that Buzzy itself is a 2,900-line product, or that a platform has no operating cost. Comparing the source code of an established application platform with a greenfield customer build would answer a different question, and not a particularly useful one.

> **The hard work did not disappear; it became inspectable much earlier.**

No customer name, production data, credentials, or confidential customer-specific details appear in this paper. The prototype uses synthetic data.

Why Pick Sales Incentive Planning?

Because it is awkward in exactly the right ways.

Published estimates vary with market definitions, but place the global Sales Performance Management market at roughly **US$2.6-2.8 billion in 2024**. [12], [13] It is served by large enterprise platforms and specialist SPM/ICM vendors, and it has accumulated its complexity honestly. A credible system must answer questions such as:

- Which sales records count, and which do not?
- Which version of a plan applied when the sale was credited?
- Was the seller above the threshold, in an accelerator tier, capped, adjusted, or clawed back?
- Who approved the adjustment?
- Can finance reconcile the result?
- Can the seller see how the number was produced without needing a minor in forensic accounting?

That is why this category makes a worthwhile stress test. It is not CRUD with a motivational quote stapled to the header. It involves changing rules, sensitive compensation information, source-system ambiguity, and an expectation that every number can be reproduced.

The Hypothesis

The useful contrast is not AI versus no AI. That argument is already over.

The real choice is between two ways of handling the application that AI helps create:

Code-first AI	Semantic application approach
AI primarily generates implementation code	AI generates and updates an application definition
Requirements become dispersed through components, routes, migrations, and services	Requirements remain visible as briefs, flows, data models, blueprints, and risk decisions
Every change can become a code investigation	Changes start in a readable business or technical artifact
The custom surface tends to grow	Custom code is reserved for specialised edges
Teams inherit the repeatable plumbing	The platform manages repeatable application infrastructure

Our hypothesis was simple: if the durable application definition remains visible and governed, AI can accelerate delivery without making future change an archaeological dig through generated code.

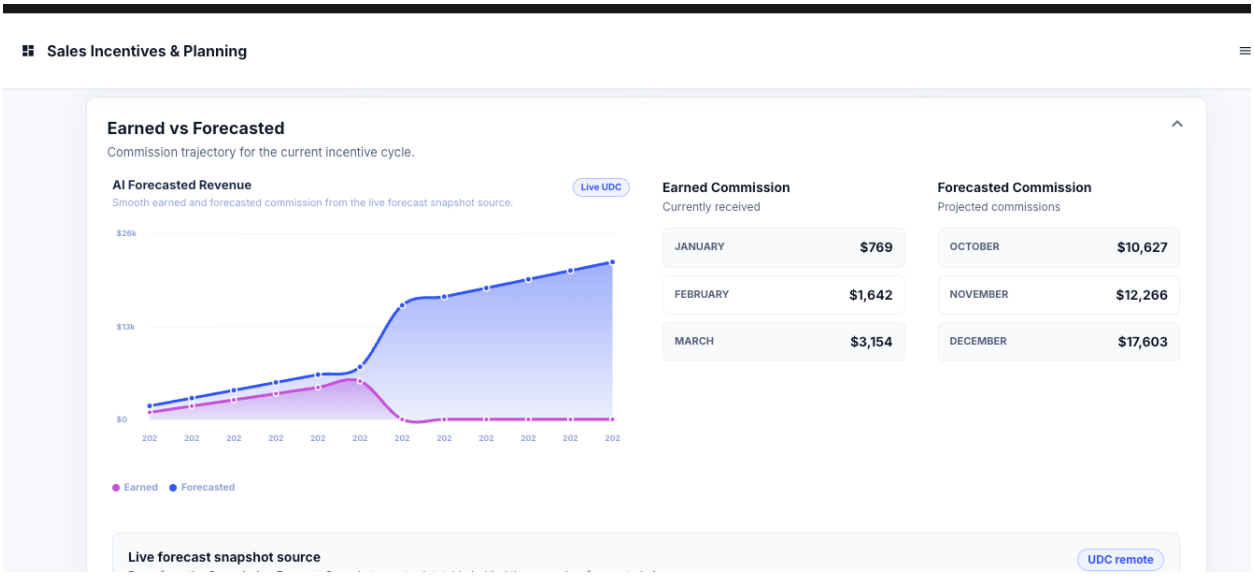
What We Built

The prototype was deliberately broad enough to expose the real seams of the category, but bounded enough to complete in days.

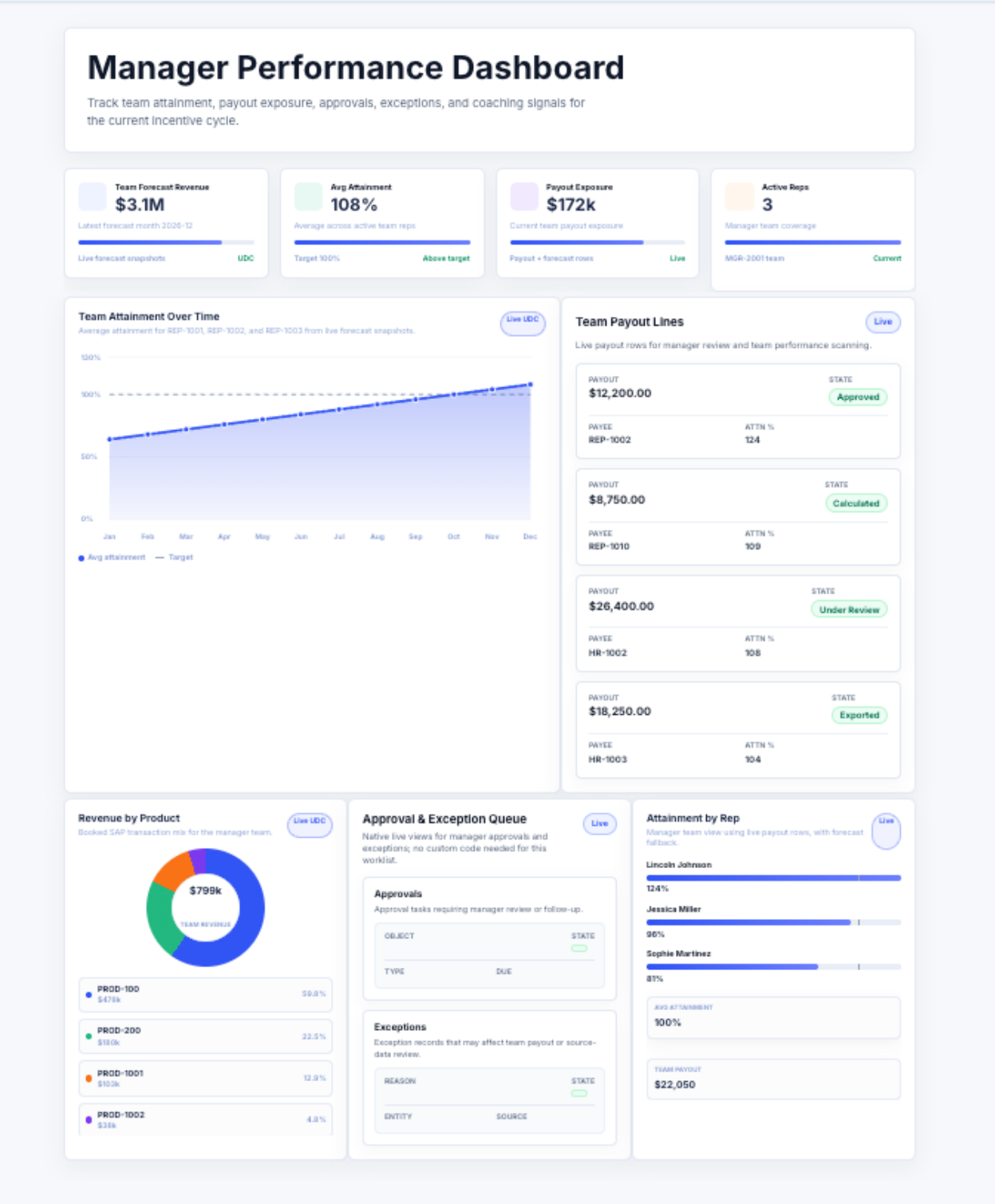
- Five role-based experiences: seller, manager, sales operations, finance, and executive.
- A control tower, planning workspace, plan and rules views, payout runs, approvals, exceptions, reconciliation, and exports.
- 29 managed and remote data tables.
- Simulated CRM, SAP, HRIS, ERP, and analytics sources.
- Seller statements and calculation waterfalls with source references and intermediate values.
- Live charts using app and remote data, rather than attractive but inert dashboard screenshots.
- Semantic artifacts for requirements, flows, data, screens, and risk decisions.

The Prototype Was Not A Wireframe

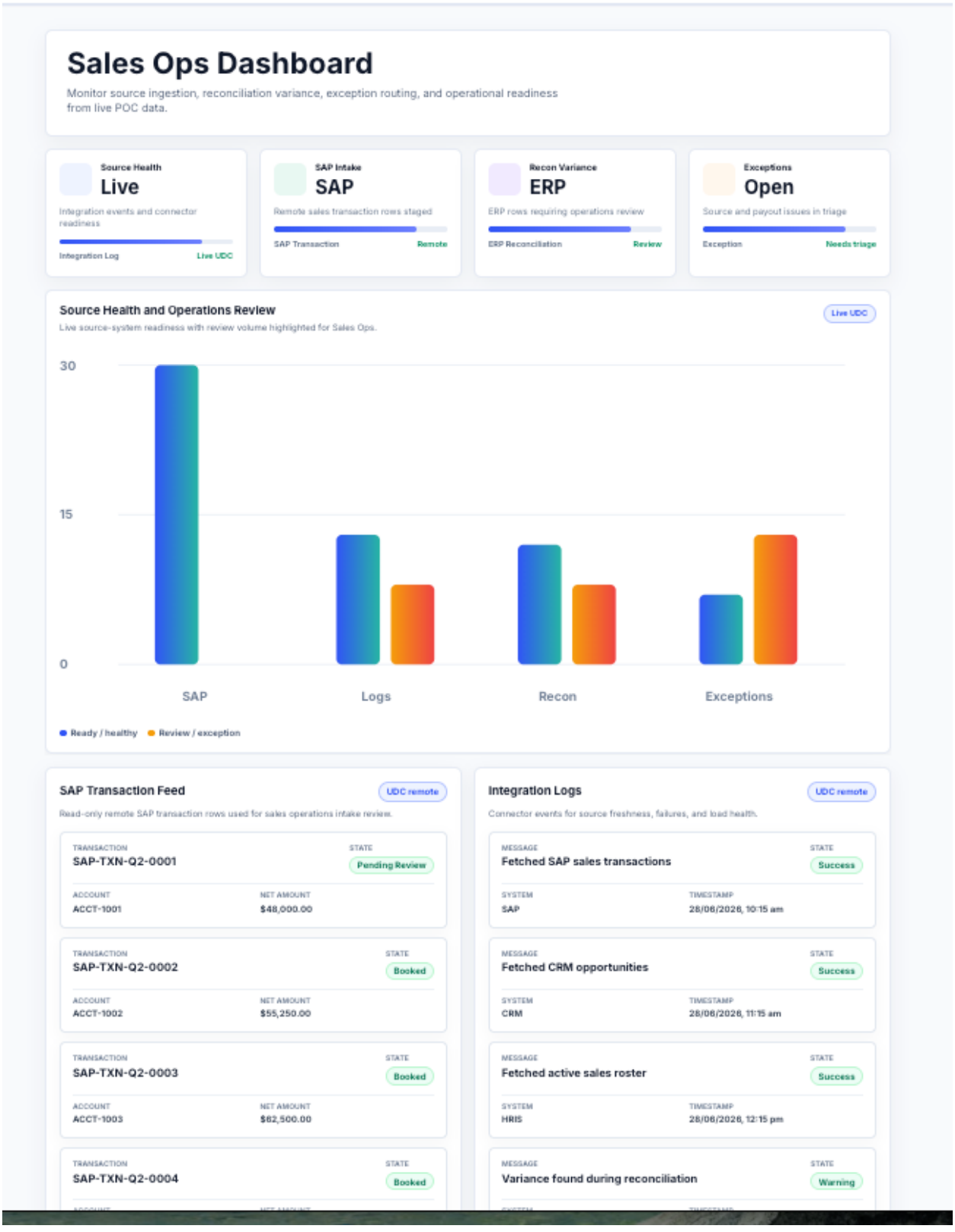
The prototype ran the same governed application model through different role contexts. Each view used the same visual system, its appropriate source and workflow data, and a deliberately restrained amount of custom code: native application components for ordinary cards and record panels, with narrowly-scoped code widgets only where chart interaction genuinely required them.



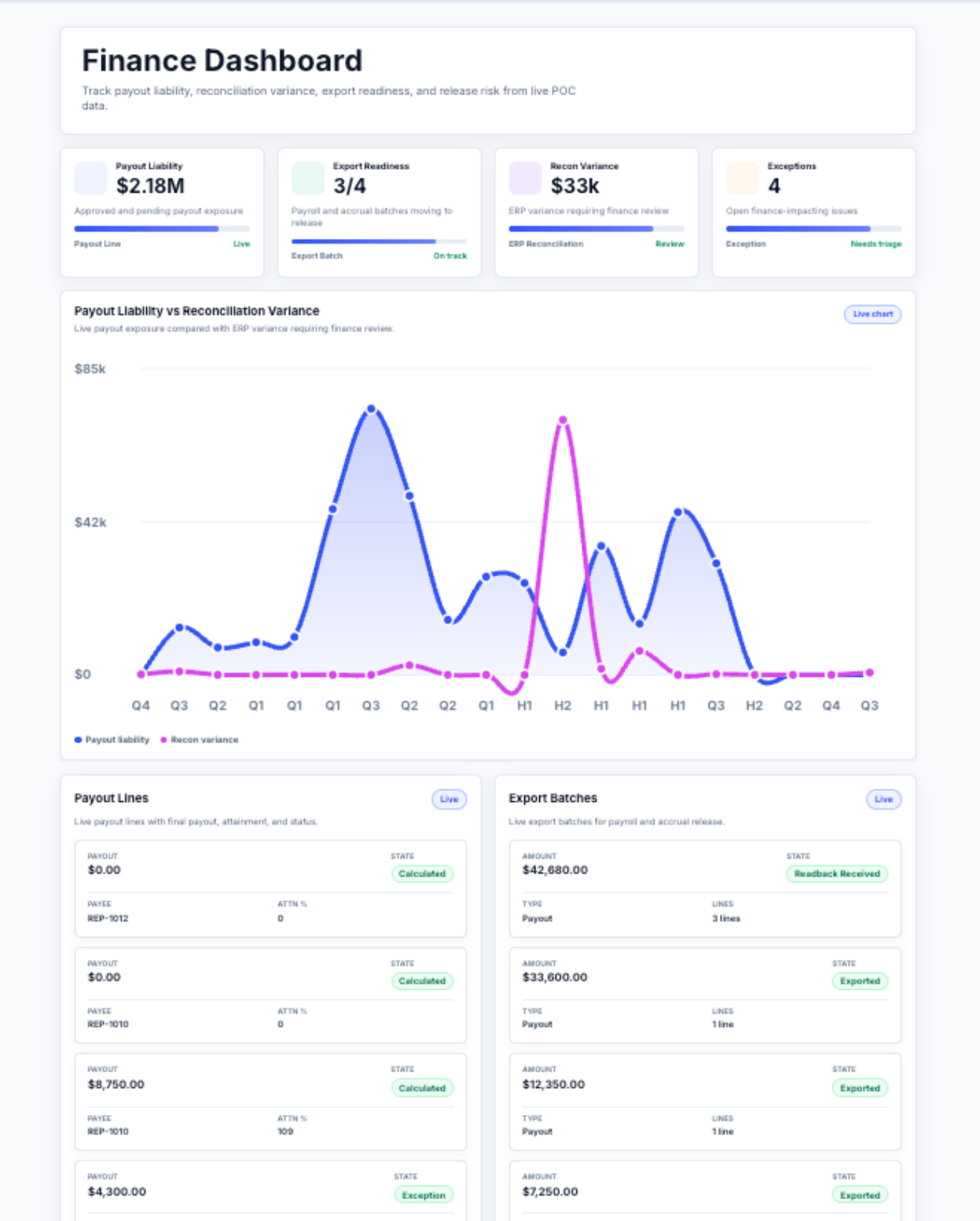
Sales Rep dashboard with live earned versus forecast commission chart



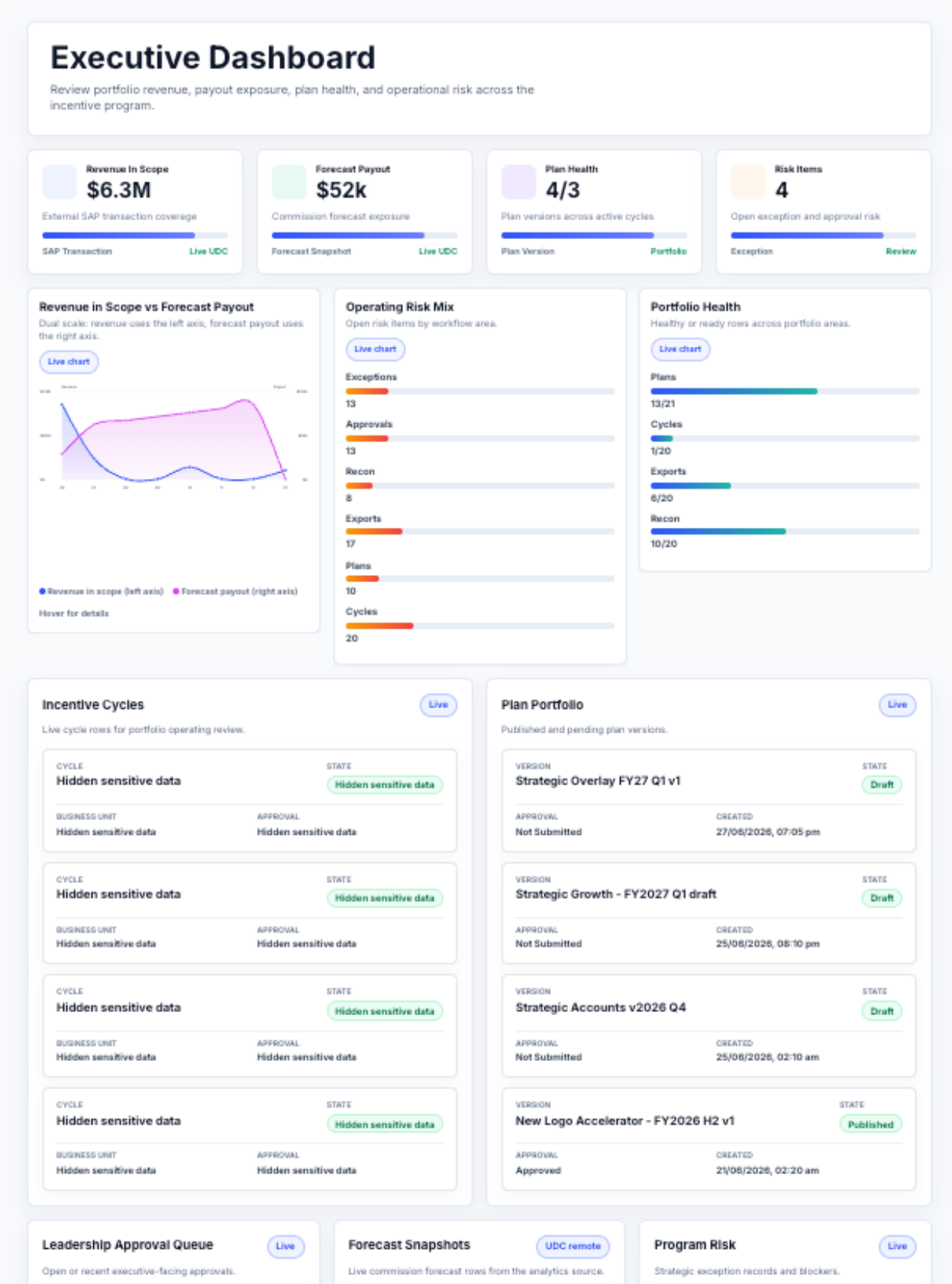
Manager Performance Dashboard showing team attainment, payout lines, revenue mix, approval queue, and attainment by rep



Sales Ops Dashboard showing source-health monitoring, SAP intake, reconciliation variance, exception routing, remote transactions, and integration logs



Finance Dashboard showing payout liability, export readiness, reconciliation variance, payout lines, and export batches


Executive Dashboard
Review portfolio revenue, payout exposure, plan health, and operational risk across the incentive program.
Revenue In Scope
$6.3M
External SAP transaction coverage
SAP Transaction
Live UDC
Forecast Payout
$52k
Commission forecast exposure
Forecast Snapshot
Live UDC
Plan Health
4/3
Plan versions across active cycles
Plan Version
Portfolio
Risk Items
4
Open exception and approval risk
Exception
Review
Revenue in Scope vs Forecast Payout
Dual scale: revenue uses the left axis, forecast payout uses the right axis.
Live chart
Revenue in scope (left axis)
Forecast payout (right axis)
Hover for details
Operating Risk Mix
Open risk items by workflow area.
Live chart
Exceptions
13
Approvals
13
Recon
8
Exports
17
Plans
10
Cycles
20
Portfolio Health
Healthy or ready rows across portfolio areas.
Live chart
Plans
13/21
Cycles
1/20
Exports
6/20
Recon
10/20
Incentive Cycles
Live
Live cycle rows for portfolio operating review.
CYCLE
Hidden sensitive data
STATE
Hidden sensitive data
BUSINESS UNIT
Hidden sensitive data
APPROVAL
Hidden sensitive data
CYCLE
Hidden sensitive data
STATE
Hidden sensitive data
BUSINESS UNIT
Hidden sensitive data
APPROVAL
Hidden sensitive data
CYCLE
Hidden sensitive data
STATE
Hidden sensitive data
BUSINESS UNIT
Hidden sensitive data
APPROVAL
Hidden sensitive data
CYCLE
Hidden sensitive data
STATE
Hidden sensitive data
BUSINESS UNIT
Hidden sensitive data
APPROVAL
Hidden sensitive data
Plan Portfolio
Live
Published and pending plan versions.
VERSION
Strategic Overlay FY27 Q1 v1
STATE
Draft
APPROVAL
Not Submitted
CREATED
27/06/2026, 07:05 pm
VERSION
Strategic Growth - FY2027 Q1 draft
STATE
Draft
APPROVAL
Not Submitted
CREATED
25/06/2026, 08:10 pm
VERSION
Strategic Accounts v2026 Q4
STATE
Draft
APPROVAL
Not Submitted
CREATED
25/06/2026, 02:10 am
VERSION
New Logo Accelerator - FY2026 H2 v1
STATE
Published
APPROVAL
Approved
CREATED
21/06/2026, 02:20 am
Leadership Approval Queue
Live
Open or recent executive-facing approvals.
Forecast Snapshots
UDC remote
Live commission forecast rows from the analytics source.
Program Risk
Live
Strategic exception records and blockers.

Executive Dashboard showing revenue in scope, forecast payout, plan health, operational risk, incentive cycles, and plan portfolio

This matters because an incentive system is not one screen. It is a connected set of role-specific operating surfaces: sellers need an explanation; managers need team performance and approvals; operations need source-health and exception visibility; finance needs liability and export readiness; leadership needs portfolio-level risk and exposure. A working prototype has to make those relationships inspectable, not merely list them in a slide deck.

The point was not to reproduce every feature of every incumbent. The point was to make the application tangible enough for sales, finance, and technical stakeholders to challenge the model before a conventional codebase hardened around the wrong assumptions.

The Number That Matters: Code Surface

The appeal of AI-generated software is obvious: it makes the first version cheap. The catch is that a first version is not the final bill.

A conventional implementation with the scope above would need code for authentication, permissions, APIs, persistence, responsive screens, workflows, integrations, calculation traceability, tests, and deployment. We estimate that a prototype-equivalent custom build would produce **50,000-129,000 lines of raw code** across those concerns.

Delivery approach	Raw custom code surface	Where most application structure lives
Conventional custom implementation	Estimated 50,000-129,000 lines	Frontend, backend, database, workflow, integration, test, and deployment code
This Buzzy prototype	Approximately 2,900 lines	Buzzy-managed briefs, flows, data model, screens, native components, permissions, and workflow artifacts

The Buzzy figure is deliberately modest rather than magical. It combines a 932-line deterministic Node/Express mock datasource with roughly 2,000 lines of narrow chart and waterfall widget markup observed in the prototype mirror. It measures the customer-specific code surface, not Buzzy's platform implementation. It does not mean there was no engineering work. It means that the majority of the application's ordinary structure did not have to become bespoke code.

The fuller methodology is in Appendix E. The main point is more important than the decimal places: the platform changes which parts of the system the team must write and carry forward.

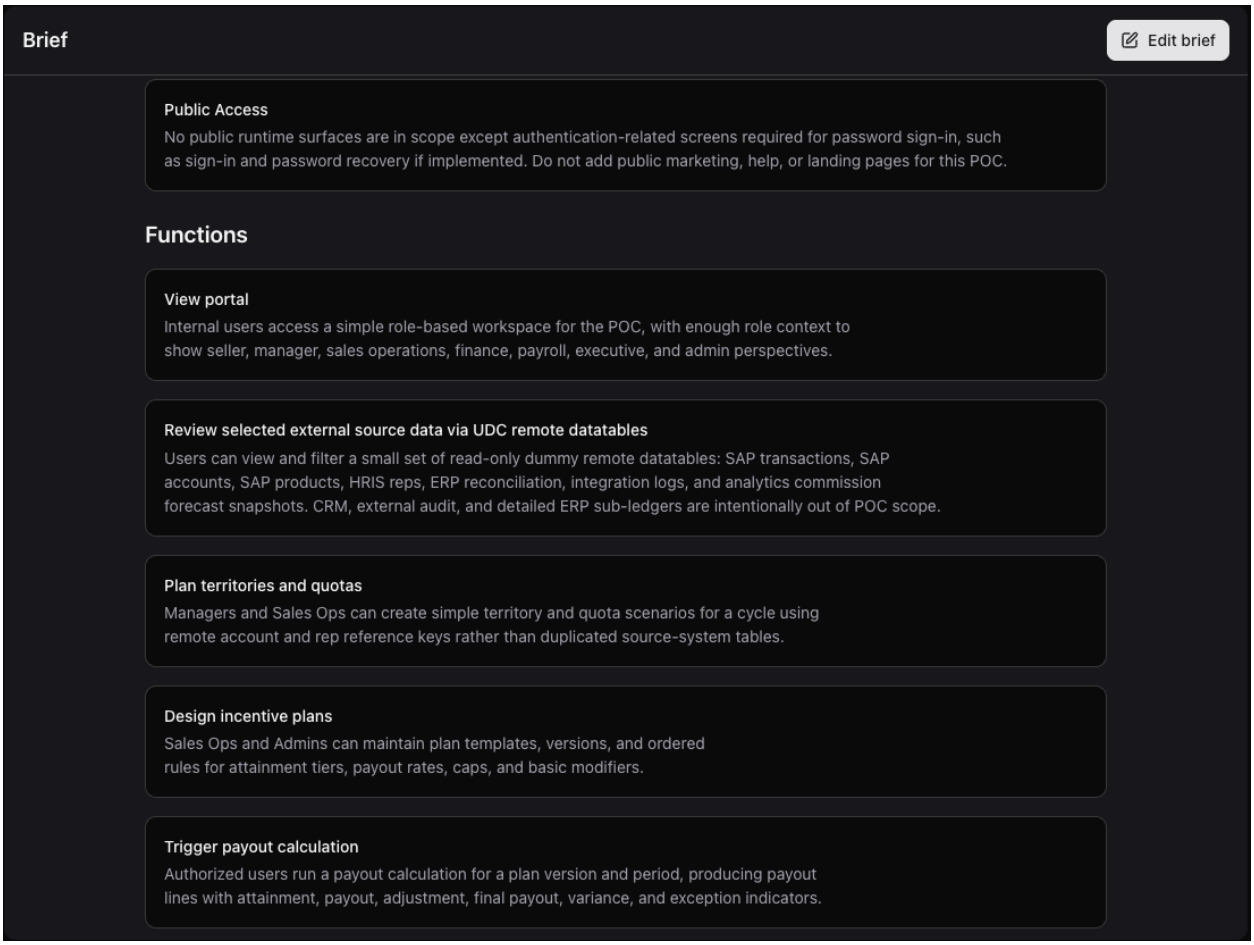
The Application Definition Is The Product, Too

Most teams can read a requirement. Fewer can safely infer that requirement from a collection of generated React components, API handlers, migrations, and half-remembered prompts. That is not an indictment of code; it is an observation about where intent goes to hide.

In this prototype, Buzzy kept the important layers visible:

- **Brief:** What are we building, for whom, and under what constraints?
- **Flow:** How does the process behave when it encounters an exception, a variance, or an approval?
- **Data model:** What state does the application own, and who can access it?
- **Blueprint:** What screens and navigation paths are required for each role?
- **Risk review:** What must be fixed, acknowledged, or blocked before release?

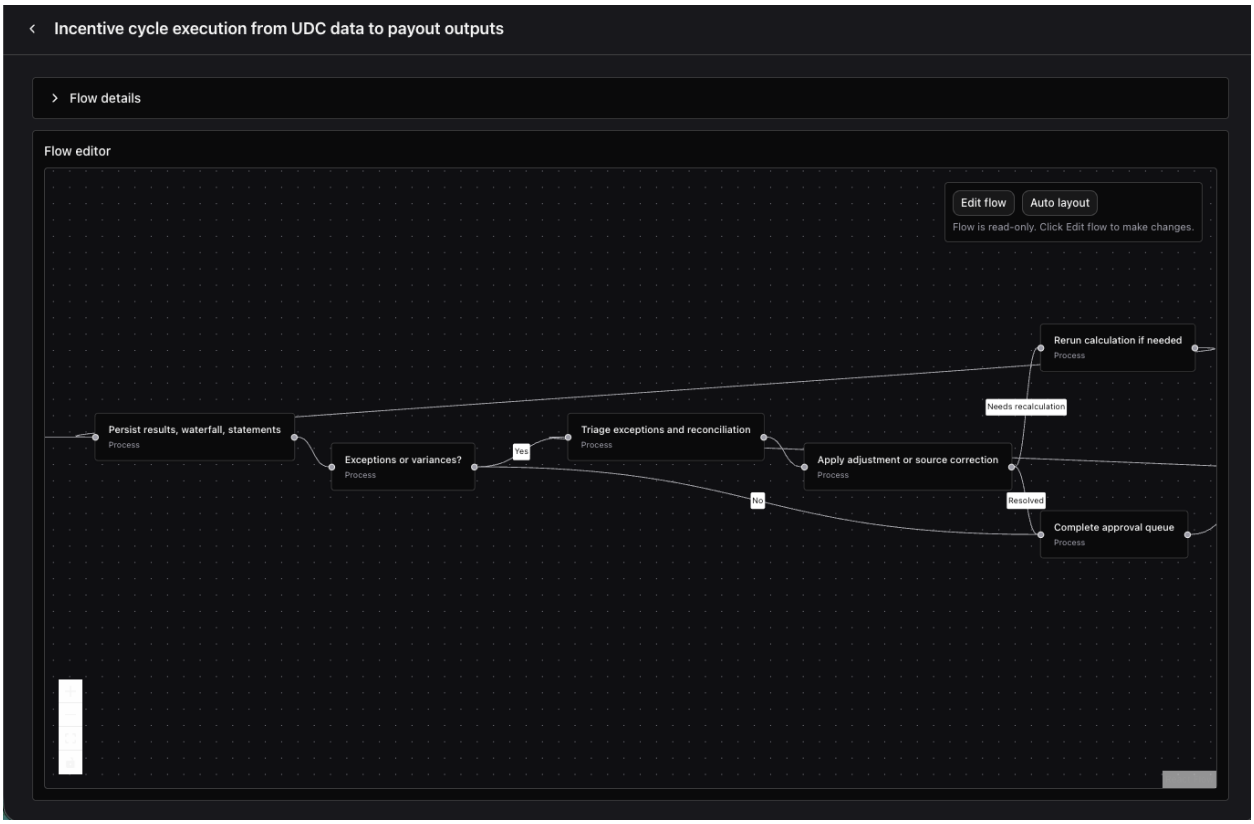
Requirements Before Routes



Buzzy brief showing role-based portal scope, source data, planning, and payout calculation requirements

The brief is where a stakeholder can argue about scope while the argument is still cheap. In this case, it captured role-based access, source data boundaries, planning scenarios, and payout outcomes.

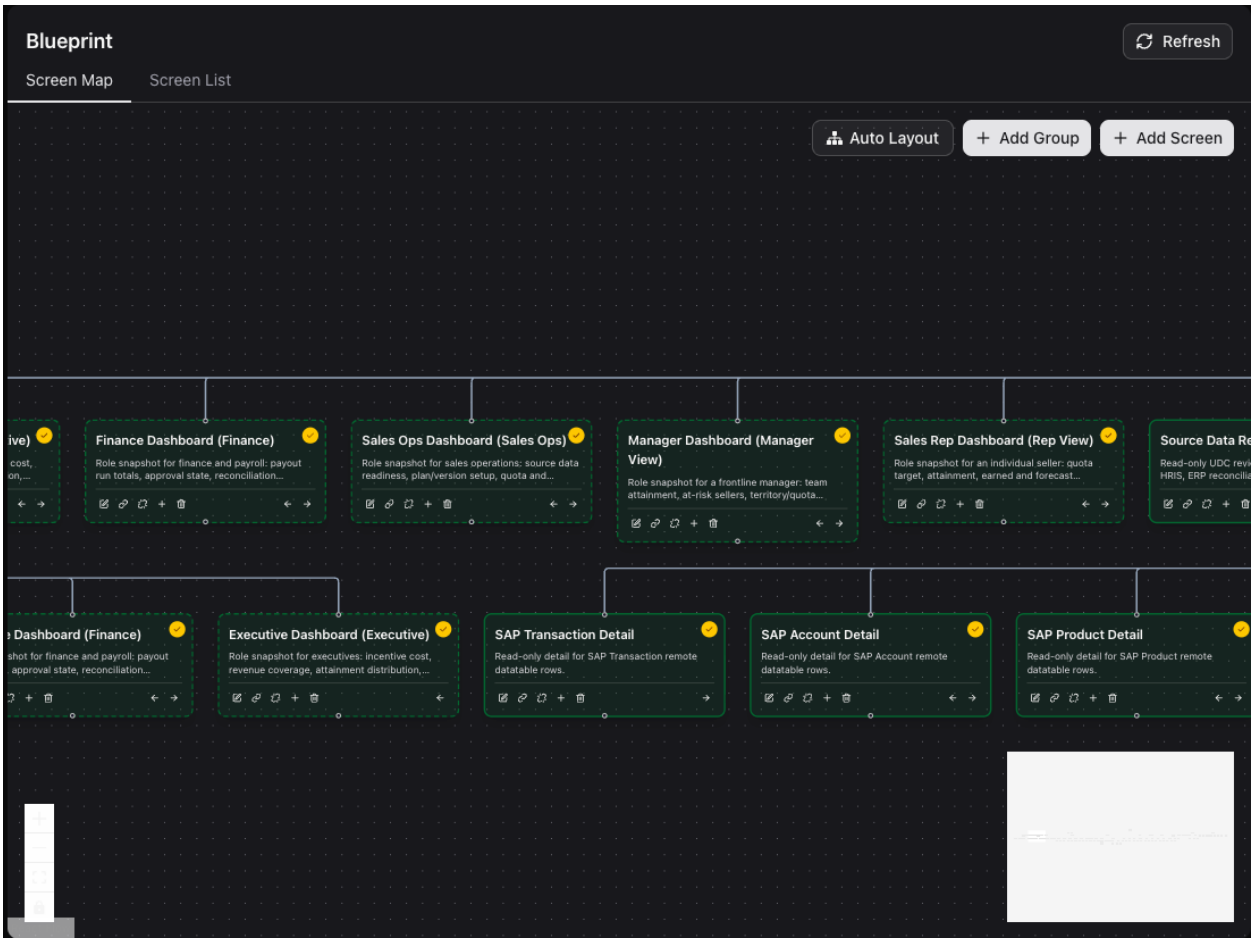
The Business Process, Not Just The Happy Path



Buzzy flow editor showing exception, reconciliation, recalculation, and approval paths

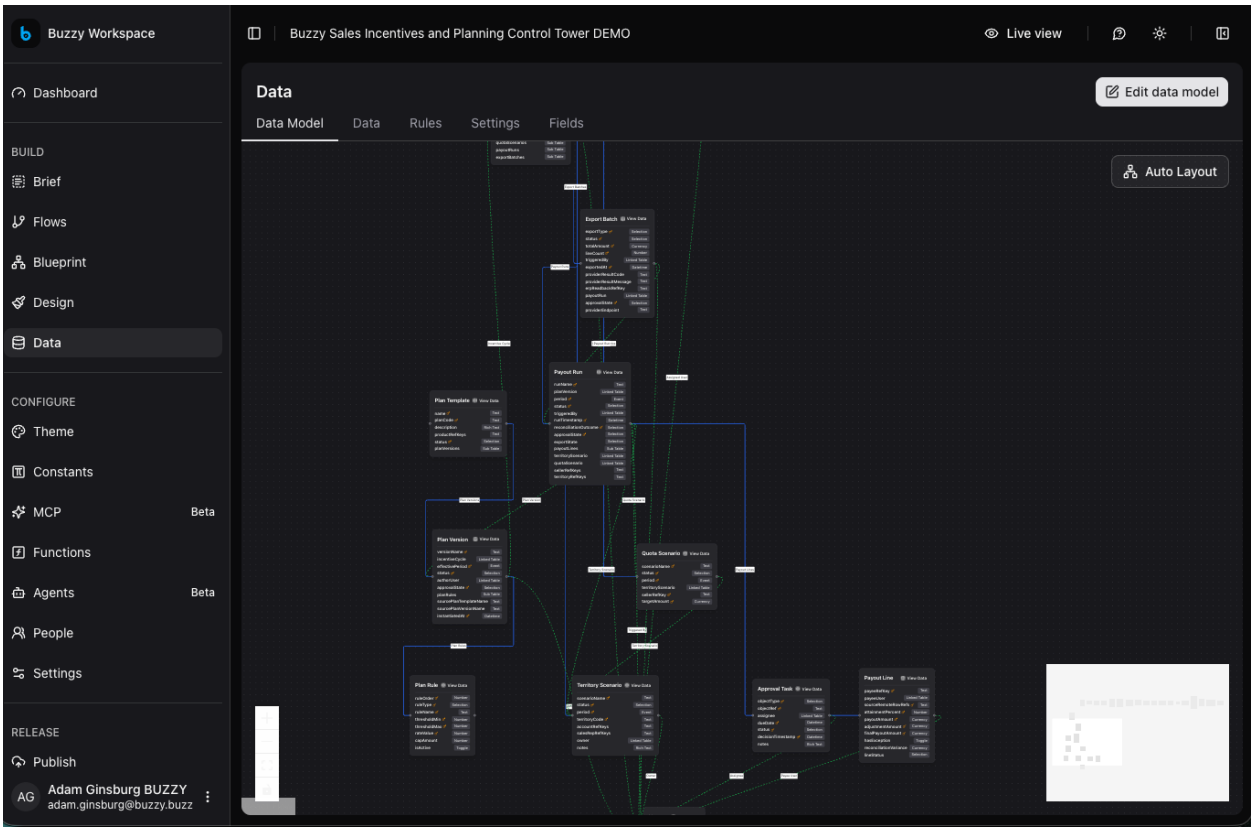
The flow makes the consequential path visible: record the result, identify a variance, triage it, apply a correction, rerun if needed, and complete the approval queue. That is what the business process looks like before it becomes a tangle of callbacks and good intentions.

A Map Of The App, And A Record Of Its Risks



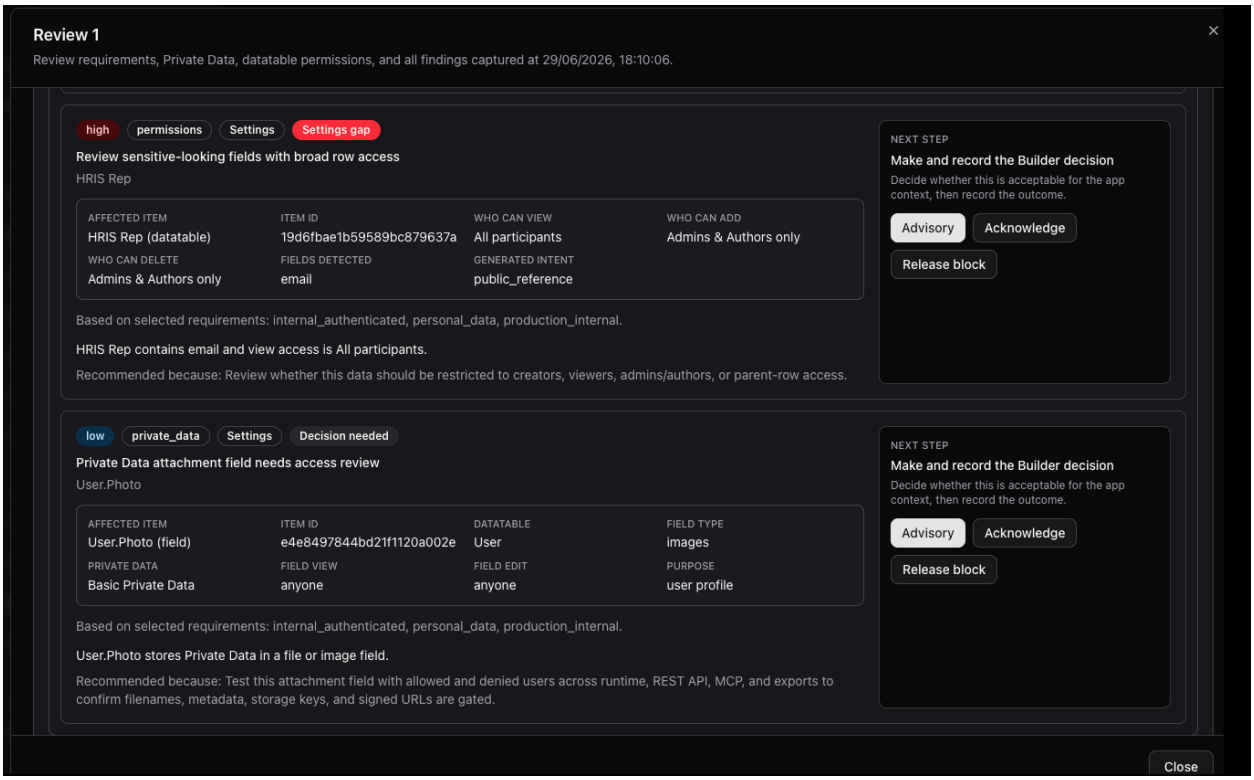
Buzzy blueprint screen map showing role views and detail screens required by the application

The blueprint gives a team a reviewable map of role views and detail screens. The corresponding data model keeps Buzzy-managed workflow state separate from remote source facts.



Buzzy data model showing the connected planning, payout, approval, and workflow entities behind the prototype

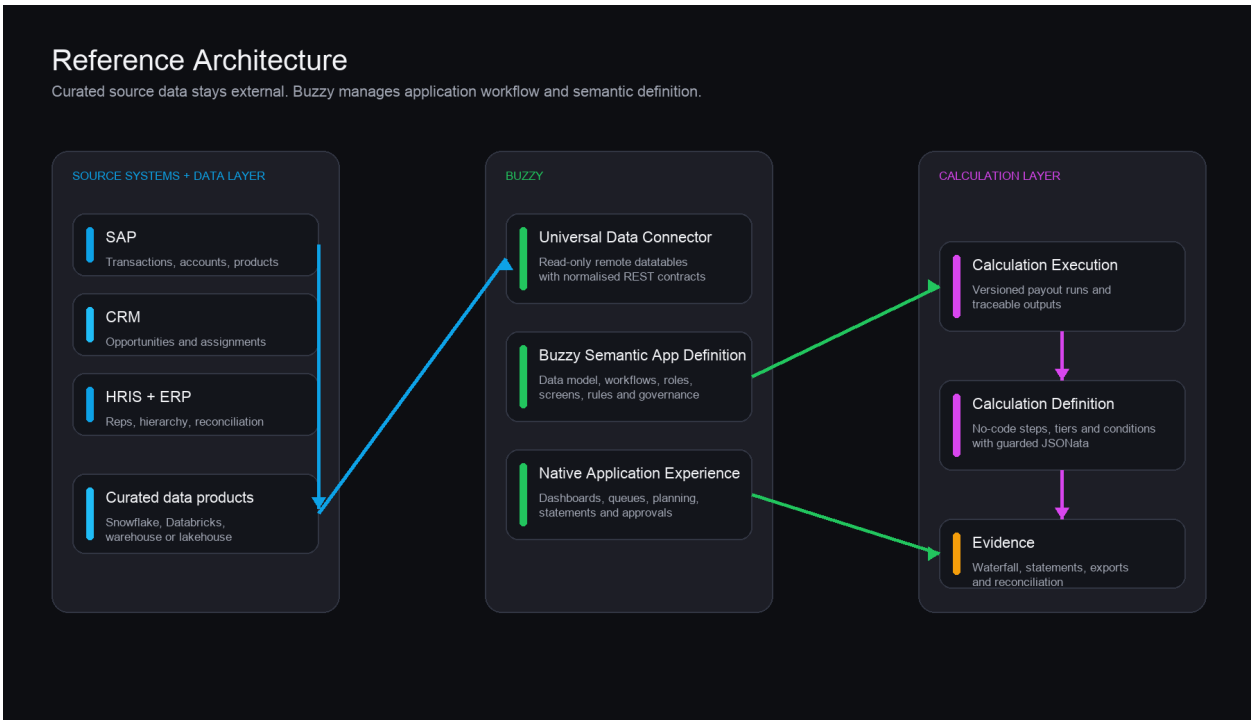
The data model is not decorative metadata. It captures the state the application owns: plan and payout lifecycle, approval tasks, exceptions, reconciliations, and calculated outputs. Remote source-system facts stay separate, which avoids copying the same enterprise data into several local tables merely to make a screen look populated.



Buzzy review findings showing permission and private-data risks with explicit decision actions

The risk review is especially relevant in compensation software. It names affected data, current access, and an explicit decision path: advisory, acknowledge, or release block. That is more useful than discovering in a security review that the HR table was visible to the entire universe.

Architecture: Do Not Put The Entire Enterprise In The App



Reference architecture showing external systems, Buzzy, and calculation responsibilities

The architecture separates three jobs:

1. **Enterprise data:** curated CRM, ERP, HRIS, sales, and financial facts. In production, these should normally arrive as governed data products through a warehouse or lakehouse, not as a festival of point-to-point integrations.

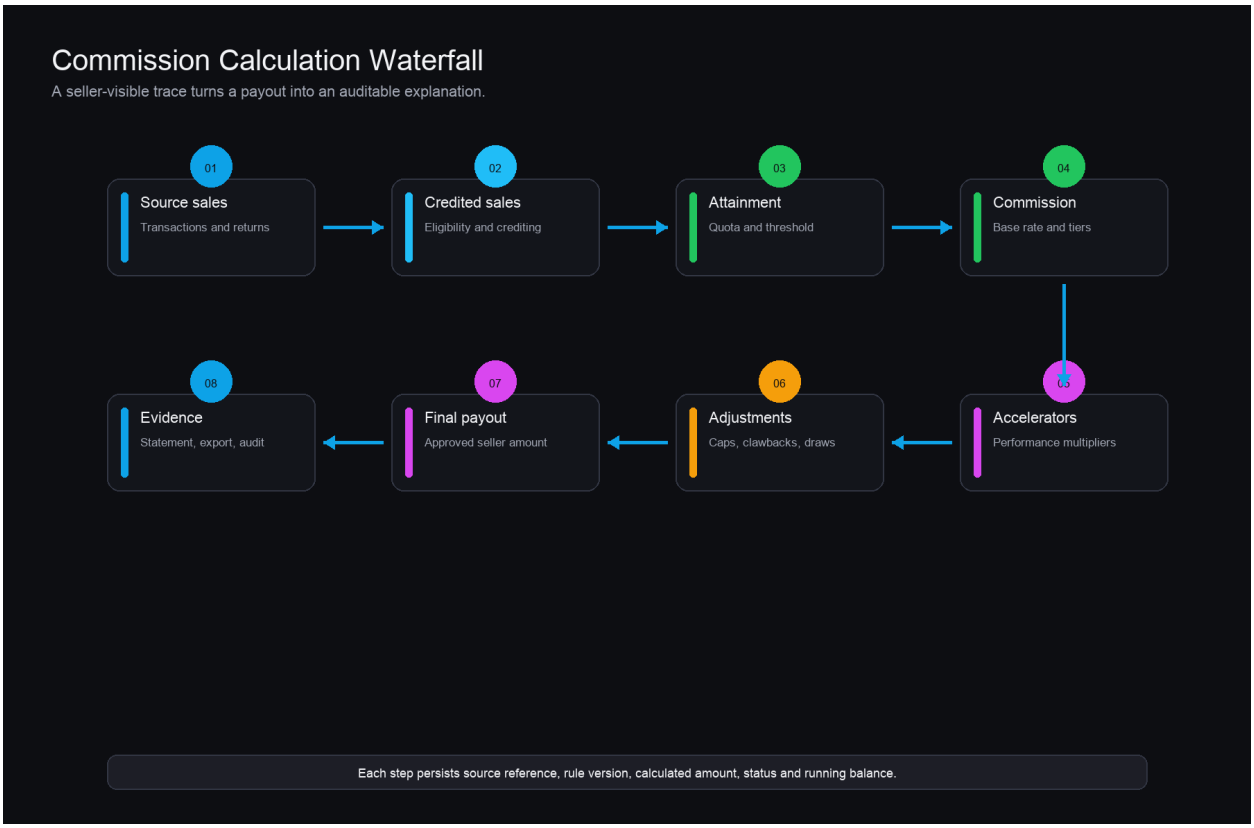
2. **Buzzy:** the application definition, user experience, workflow state, permissions, and governance.

3. **Specialised execution:** bounded services such as calculation execution, where specialised logic and integration contracts genuinely deserve code.

This is not an attempt to replace SAP, a data warehouse, or payroll. It is an attempt to stop every new operational workflow from demanding a brand-new hand-built application shell.

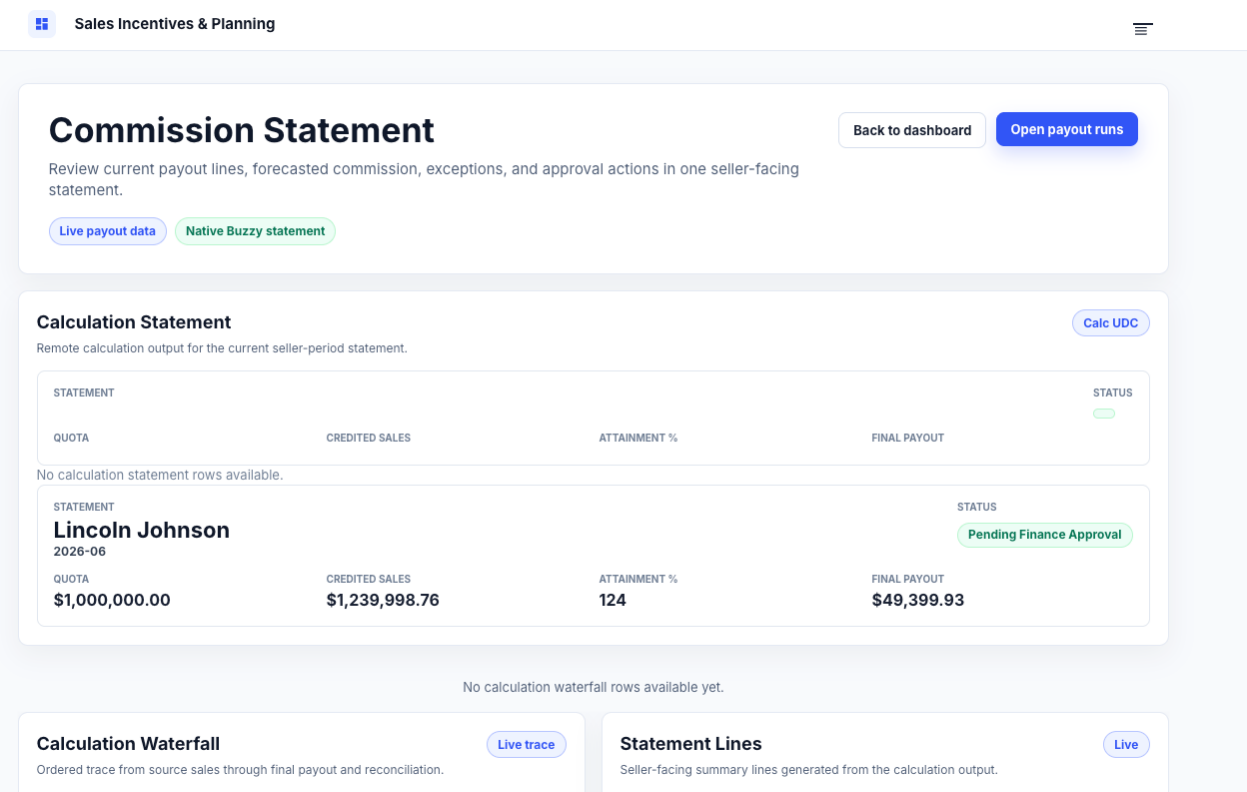
Where The Money Comes From - And Why That Is Hard

The most consequential screen in an incentive system is not the executive dashboard. It is the page that explains a seller's payout.



Commission calculation waterfall from source sales through auditable payout evidence

For the prototype, a payout was treated as evidence rather than a mysterious number that had drifted out of a black box. The calculation trace captures source facts, calculation version, rule or formula, intermediate amount, adjustment, approval, and final result.



Seller-facing Commission Statement showing quota, credited sales, attainment, final payout, calculation evidence, and statement lines

The seller-facing Commission Statement is the practical expression of that trace. It puts the payout, quota, credited sales, attainment, approval status, calculation waterfall, and statement lines in one place. A seller does not need a finance analyst to translate an unexplained total; they can see the evidence behind it, while finance retains the underlying approval and reconciliation trail.

That is the right direction for a future no-code calculation builder:

- calculation definitions are versioned, reviewed, tested, and approved;
- source facts remain external and read-only;
- execution persists results and trace rows;
- dashboards consume results rather than hiding business logic in widgets; and
- advanced expressions are bounded, validated, and auditable.

In other words, a commission calculation should be able to survive a finance meeting. This is a higher bar than merely returning a number.

What The Experiment Actually Proved

It proved that a complex enterprise scope can be translated into a working, inspectable system quickly. Stakeholders could challenge the data model, the exception path, the permissions, and the calculation story while those things were still explicit artifacts.

It also showed that custom code can be concentrated. The synthetic data service, specialised charting, and calculation work remained code. The bulk of the application did not.

That distinction matters because AI adoption is already associated with greater delivery throughput, while high-quality internal platforms remain foundational to getting value from it. [11] Faster code creation is not the same as faster, safer, or more maintainable delivery.

What It Did Not Prove

It did not prove production-scale performance, payroll accuracy, full security hardening, or lower total cost of ownership over several years. It did not replace a mature SPM suite. Any claim otherwise would be the sort of demo bravado that makes security teams reach for the decaf.

The right next step is a bounded pilot, not a press release about the end of enterprise software.

The Conclusion: SaaS Is Not Dead; Opaque Code Piles Are Not A Great Alternative

The SaaSpocalypse may be real in the sense that many categories of software can now be challenged far more quickly than before. A team can build around its own process instead of accepting every assumption embedded in a packaged product.

But replacing SaaS with a vast, opaque body of generated code simply swaps one dependency for another. The subscription invoice may disappear; the maintenance backlog, audit burden, and security responsibility do not.

The more credible future is AI plus a governed semantic application definition plus a small number of bounded custom services. Let AI accelerate analysis and construction. Keep the model, workflow, rules, permissions, and risk decisions visible. Write code where code is genuinely the right tool, and resist making it the default storage format for every business decision.

That is a less cinematic ending than the death of SaaS. It is also more useful.

A Practical Next Step

For organisations that want to test the approach, the sensible path is staged:

1. **Discovery and prototype:** define the semantic model, build one narrow workflow, and validate the experience.

2. **Historical shadow run:** compare outputs against sanitised historical data and explain the variances.

3. **Controlled production pilot:** harden access, audit, observability, integration, and operational controls for a defined workflow.

Buzzy is open to scoped services engagements for teams that want to apply this approach to a real use case. Start with a bounded problem, agreed data boundaries, controls, and success criteria. Reach out through Buzzy.

Appendix A: Market And Category Context

Sales Performance Management and Incentive Compensation Management products administer commission-based plans for sellers and revenue producers. Typical capabilities include plan design, calculation and payout, territory and quota management, statement visibility, workflow, and integration with CRM, ERP, and HCM systems. [1]

Representative products include Varicent Incentives, Xactly Incent, Salesforce Spiff, CaptivateIQ, Performio, SAP SuccessFactors Sales Performance Management, Oracle Sales Performance Management, Anaplan for Sales, Everstage, beqom, and QCommission.

Pricing is typically quote-based for enterprise SPM/ICM. Salesforce Spiff publicly lists a US$75-per-user-per-month annual-billing price for Incentive Compensation Management; Xactly and CaptivateIQ describe pricing as dependent on payees, plan complexity, integrations, and services. [2][3][4] The price is only part of the bill: implementations also involve integration, historical reconciliation, plan-rule validation, training, and ongoing change support.

Appendix B: Prototype Scope

The prototype included Buzzy-managed workflow tables for cycles, templates, plan versions, rules, scenarios, payout runs and lines, exceptions, approvals, and exports. Remote UDC-style tables represented source-system facts from sales, CRM, HRIS, ERP, and analytics services. Calculation output tables represented runs, results, waterfalls, seller statements, adjustments, clawbacks, and disputes.

The main experiences covered sales representative, manager, sales operations, finance, executive, and administrative perspectives. This was intentionally a proof of concept, not a complete model of every enterprise SPM variant.

Appendix C: Synthetic Integration Design

The prototype used a small Node/Express synthetic datasource to simulate SAP, CRM, HRIS, ERP, and analytics feeds. It returned stable, normalised list responses that Buzzy remote datatables could query and refresh.

For production, direct application-to-application endpoints should normally be replaced or supplemented by curated, versioned data products in a warehouse or lakehouse. That reduces point-to-point integration code and provides a more stable contract for calculation, reporting, and reconciliation.

Appendix D: Calculation Design Notes

The calculation proof of concept covered gross-to-net sales, eligibility, crediting, quota attainment, thresholds, commission rates, accelerators, caps, adjustments, clawbacks, draw recovery, final payout, and incumbent-system variance.

The future calculation builder should manage templates, versions, steps, conditions, tiers, input mappings, test cases, and publication state inside Buzzy. JSONata or a comparable expression language can support advanced step-level logic, provided it is bounded, versioned, validated, and auditable.

Appendix E: Code-Surface Comparison And Methodology

The 50,000-129,000-line conventional range is a directional prototype-equivalent estimate, not a claim that every implementation would have exactly this size. It covers the code surface that a conventional team would normally own for the breadth demonstrated here:

Area	Conventional custom-code estimate
Data model, APIs, validation, and persistence	8,000-18,000 lines
Authentication, roles, permissions, and routing	3,000-8,000 lines
Responsive dashboard and workflow screens	20,000-45,000 lines
Charts and visualisations	3,000-8,000 lines

Integration adapters and synthetic backends	3,000-10,000 lines
Calculation engine and traceability	5,000-20,000 lines
Tests, fixtures, deployment, and operations scripts	8,000-20,000 lines
Total prototype-equivalent range	**50,000-129,000 lines**

For the Buzzy prototype, the observable customer-specific custom surface was approximately 2,900 lines: 932 lines in the deterministic Node/Express datasource and roughly 2,000 lines of narrow chart/waterfall widget markup in the local mirror. Some widget variants were duplicated during iteration, so this is a rounded comparison rather than a precision claim. It does not count Buzzy's own platform implementation; the comparison is between the customer-specific code the team would own in either delivery approach.

The meaningful distinction is architectural. Buzzy represented the broad application surface - requirements, flows, data model, screens, navigation, native components, permissions, and workflow state - as managed semantic artifacts rather than custom frontend and backend code. The long-term proposition still needs to be tested through real change cost, control quality, and operational performance.

Sources

- [1] Gartner Peer Insights, "Sales Performance Management Reviews and Ratings", accessed July 2026. https://www.gartner.com/reviews/market/sales-performance-management
- [2] Salesforce, "Salesforce Spiff Pricing", accessed July 2026. https://www.salesforce.com/sales/incentive-compensation-management/pricing/
- [3] Xactly, "Pricing", accessed July 2026. https://www.xactlycorp.com/pricing
- [4] CaptivateIQ, "Pricing", accessed July 2026. https://www.captivateiq.com/pricing
- [5] G2, "Varicent Pricing", accessed July 2026. https://www.g2.com/products/ibm-sales-performance-management/pricing
- [6] Varicent, "Solution Services", accessed July 2026. https://www.varicent.com/solutions/services
- [7] Performio, "Sales Commission & Incentive Compensation Software", accessed July 2026. https://www.performio.co/

- [8] National Institute of Standards and Technology, "SP 800-218: Secure Software Development Framework (SSDF) Version 1.1", accessed July 2026. https://csrc.nist.gov/pubs/sp/800/218/final
- [9] OWASP Foundation, "Top 10 for Large Language Model Applications", accessed July 2026. https://owasp.org/www-project-top-10-for-large-language-model-applications/
- [10] Buzzy, "The Hidden Cost of AI Application Delivery", accessed July 2026. https://www.buzzy.buzz/whitepapers/hidden-cost-ai-application-delivery/
- [11] Google Cloud / DORA, "State of AI-Assisted Software Development 2025", accessed July 2026. https://dora.dev/research/2025/dora-report/
- [12] Grand View Research, "Sales Performance Management (SPM) Market Size & Outlook, 2030", accessed July 2026. https://www.grandviewresearch.com/horizon/outlook/sales-performance-management-spm-market-size/global
- [13] Global Industry Analysts, "Sales Performance Management", accessed July 2026. https://www.marketresearch.com/Global-Industry-Analysts-v1039/Sales-Performance-Management-42600596/