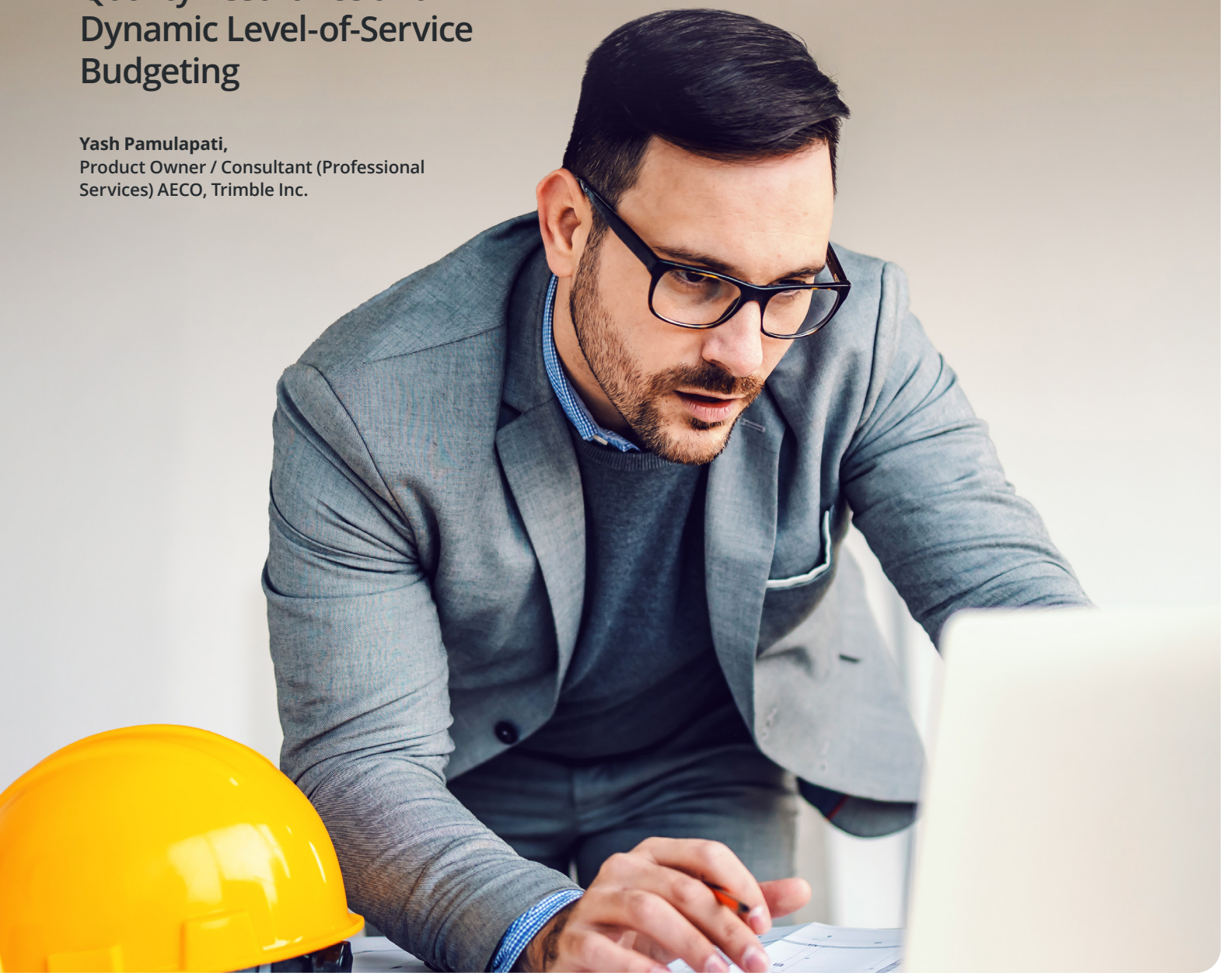
Trimble

From Spreadsheet to System:

Automating Maintenance Quality Assurance and Dynamic Level-of-Service Budgeting

Yash Pamulapati,
Product Owner / Consultant (Professional Services) AECO, Trimble Inc.



Executive Summary

For decades, state departments of transportation (DOTs) have managed sprawling roadway networks — tens of thousands of lane miles and thousands of major structures — on the strength of institutional knowledge and a lattice of interlocking spreadsheets. As Maintenance Quality Assurance (MQA) programs matured and field-survey data accumulated over a decade or more, the spreadsheet model began to buckle. A single workbook cannot reconcile ten-plus years of condition surveys, district-specific unit costs, localized inflation, and the combinatorial explosion of "what-if" budget scenarios across every performance measure and every district.

This article describes — at an architectural and software-engineering level — how a modern, cloud-native Asset Management System (AMS) closes the gap between the field and the finance office. It traces the journey of one major statewide transportation authority that migrated its MQA budget model out of Excel and into a dynamic **Desired Level-of-Service (LOS) engine**: a workflow in which a maintenance supervisor sets a target letter grade (A+ through F), a number of years to reach it, and an inflation assumption, and the system instantly returns a defensible, multi-year activity budget — decomposed all the way down to the individual maintenance crew.

Crucially, this was not a transformation that the software performed on its own. A platform supplies the raw capability — but capability is roughly 20% of the outcome. The remaining 80% is the work of translation: understanding the agency's legacy workflows, mapping its localized economics, and shaping a generic platform into a tool that fits one specific operational reality. That work was led by a dedicated **Product Owner** — the role that, throughout this engagement, acted as the bridge between the authority's business needs and the platform's configurability, owning the technical design, the product backlog, and the day-to-day collaboration with the core development team.

The payoff was not merely convenience. By replacing static estimates with a data-driven, condition-linked budgeting framework, one statewide authority — engaged with the platform provider over a multi-year implementation — was able to objectively justify a roughly **33% expansion in its maintenance budget** (an increase from approximately $162M to $216M), backed by hard performance data rather than narrative. That result was a joint victory — robust software guided by expert consulting. This is the story of the architecture, and of the people, that made it possible.

SECTION I

The Legacy Dilemma: Moving Beyond the Spreatdsheet

The modern DOT operates at a scale that is fundamentally hostile to manual analysis. Consider a representative statewide network: on the order of 30,000 lane miles of roadway, 6,800 major structures, and more than a decade of accumulated maintenance operations data. Each of those assets degrades on its own curve, is serviced by district crews with different labor and material costs, and contributes to a public-facing obligation: safety, mobility, and stewardship of taxpayer capital.

The spreadsheet was a rational first tool. It is universal, auditable at a glance, and infinitely flexible. But flexibility is exactly what defeats it at scale. Several structural failures emerge predictably:

- **Combinatorial blow-up.** A budget model that must evaluate N performance measures × D districts × F functional classes × Y target-year horizons quickly produces hundreds of thousands of interdependent cells. One authority maintained a separate workbook per district — guaranteeing that a methodology change had to be re-implemented by hand, district by district, with no enforcement of consistency.
- **No single source of truth.** Condition surveys lived in one system, work-order costs in another, and the forecast model in a third (Excel). Reconciliation was a manual, error-prone, annual fire drill.
- **Localized economics are invisible.** Unit costs vary by district and by year, and inflation compounds differently across a multi-year plan. Hard-coding these into formulas makes the model brittle and opaque.
- **No condition feedback loop.** A spreadsheet forecast cannot easily answer the central policy question: “If I spend X more dollars, what letter grade does the public actually get next year — and the year after?”

Recognizing *why* the spreadsheet failed is itself a consulting deliverable, not a given. Before a single field was configured, the Product Owner spent time embedded with the agency — reverse-engineering the per-district workbooks, documenting the implicit business rules buried in their formulas, and reconciling the terminology the agency used with the data structures the platform expected. Moving off spreadsheets is far less a data migration than a process-mapping exercise: the legacy logic has to be understood before it can be re-expressed in a system.

The first step out of this dilemma is **standardization**, and that is precisely what a formal MQA program provides. Launched as a deliberate initiative — and shaped, in this case, with direct guidance from the consultant on what the platform could enforce — MQA introduces three disciplines that the spreadsheet never enforced:

1. **Defined Performance Measures.** A formal catalog of "defect elements" — e.g., *Bridge: Cleaning & Sweeping (% Deficient Sq Ft), Drainage: Culvert Cleaning (% Deficient), Pavement: Asphalt Surface Defects (Deficient Sq Ft / Lane Mile)* — each tied to a major asset type and a unit of measure.
2. **A structured A–F grading scale.** Each performance measure is scored on a **thirteen-point letter scale (A+, A, A-, B+ … D-, F)**, where each grade maps to an explicit numeric band of the percent-deficient score. This converts raw survey numbers into a language executives, legislators, and the public all understand.
3. **Accountable data collection.** A formal "Data Collection Guide," trained survey crews, and a repeatable sampling process turn anecdote into a defensible, year-over-year dataset.

Standardization is the precondition for automation. Once condition is expressed as a graded, consistent measure across the entire network, software can finally do what the spreadsheet could not: treat the budget as a *function of desired condition.*

SECTION II

Engineering the Solution: The Architecture of a Dynamic LOS Engine

The heart of the platform is a layered separation between setup (configuration), data summarization (aggregation jobs), and analysis (the budgeting engine). This separation is what allows the same engine to serve any agency without code forks — the business rules live in configuration and pluggable scripts, not in hard-coded logic.

It is worth being precise about authorship here. These layers did not configure themselves. Each represents a deliberate architectural decision made by the Product Owner: deciding what belonged in configuration versus custom code, how the agency's per-district Excel economics would be normalized into a scalable design, and how to sequence the work so the core development team could build the genuinely new capabilities while the consultant configured the rest. The platform provided the building blocks; the consultant designed the building.

2.1 The configuration layer: mapping condition to money

Three setup constructs define the entire model — and each required the consultant to translate a fuzzy business intent into an exact, machine-enforceable rule:

- **Performance Measures.** Each measure carries not just a name and asset type, but an *embedded SQL expression* that calculates the LOS score per district and fiscal year directly from the raw MQA defect-survey data. Authoring those expressions — one family per asset category, dozens in total — was a core consulting deliverable: the SME translated the agency's survey methodology into queries the platform could execute. Critically, the cost model itself is named here too — a pluggable script reference — so the agency's bespoke budgeting math is a *configuration value*, not a code change. Conversion factors, correction factors, and inventory units of measure round out the definition.
- **LOS Scale.** The 13-point scale is data, not code. Each grade is defined by lower- and upper-bound expressions (e.g., >= 0% and <= 1.67% A+). Because the grade boundaries are stored, a policy decision to tighten standards is a configuration edit, not a software release — a flexibility the consultant deliberately preserved so the agency could evolve its own standards independently.
- **Activity-to-Measure Mapping.** The pivotal link. Each maintenance activity (e.g., *Sweeping, Crack Pouring, Overlay*) is mapped to one or more performance measures, with a **`Percent Applicable`** weight (0–100) that apportions an activity's cost and accomplishment across the measures it serves. The platform enforces an invariant — the sum of `Percent Applicable` across measures for any activity may not exceed 100% — at save time, in the data layer. Defining this mapping demanded domain judgment, not data entry: deciding *which* crew activities legitimately move *which* condition measure is exactly the kind of decision an SME is hired to make.

2.2 The aggregation layer: pre-computing at scale

The single most important architectural decision for handling 10+ years of records is to **never make the user's UI session do heavy lifting at query time** — a decision the consultant made early, having seen how interactive queries against a decade of work orders would otherwise stall a browser session. Instead, scheduled system jobs pre-summarize the three pillars of the analysis into purpose-built summary tables:

Job (conceptual)	Produces	Grain
Fill Defect Survey Summary	LOS score & grade per measure	District × Survey Year × Functional Class
Fill Work-Order Summary by Element (run per month of the fiscal year)	Raw work-order cost & accomplishment	Element-level monthly rollup
Fill Activity Summary for Analysis	Activity cost & accomplishment, plus a derived state-level unit cost	District × Admin Unit × Activity × Year

By the time a planner opens the budgeting window, the expensive joins across a decade of work orders and surveys are already materialized. The interactive engine then operates on compact, indexed summary tables — which is what keeps the experience responsive rather than timing out a browser session. Validating these jobs against the agency's historical data, and reconciling their output back to the legacy spreadsheets the agency trusted, was a significant share of the engagement's effort — and the step that earned the agency's confidence in the new numbers.

2.3 The analysis layer: how an input becomes a budget

When the planner sets the levers — **Target LOS, Years to Target LOS, and Inflation Rate** — the engine recalculates a multi-year forecast (Scenario Cost Year 1, plus projected LOS grades for Years 1 through 5) at the *Performance Measure* × *District* × *Functional Class* grain.

This is where **pluggable business logic** earns its place — and where the consulting role is most visible. The agency's original Excel budgeting math — one workbook per district — was translated into a single, server-side **scripted cost model** (a "Models"-type script attached to each performance measure). That translation was authored and owned by the Product Owner, who reduced dozens of district-specific spreadsheets to one parameterized model that produced the same answers the agency already trusted, then defended that equivalence line by line. Embedding the calculation as an interpreted script rather than compiled platform code yields two engineering benefits:

1. **Dynamic, per-agency logic without a release.** The model can be tuned, versioned, and swapped in configuration. The same engine serves a measure whose math came from a client spreadsheet and a measure using a standard model.
2. **Server-side execution against summary tables.** The heavy iteration runs in the data tier, returning only the computed result set to the UI — so a statewide, all-districts scenario does not crash or freeze the client.

The output then drives a budget **decomposition** that pushes the district-level number down to each crew. The decomposition logic is deliberately compact and reads directly from the activity summary table:

```
public void onRetrieve(DataStore ds, Integer effYear, Integer defectId,
                       Integer ownerNet, Integer functClass, Double coef) {
    if (coef == null || coef == 0) return;            // no scenario, nothing to
decompose
    ds.retrieve(defectId, ownerNet, functClass, effYear);
    for (int i = 1; i <= ds.rowCount(); i++) {
        // Historical spend attributable to this measure
        ds.setItemNumber(i, "OLD_DEF_RELATED_BUDGET",
                         ds.getItemNumber(i, "EFF_BUDGET") * ds.getItemNumber(i, "PCT_
APPL"));
        // Projected spend = historical x applicability x scenario coefficient
        ds.setItemNumber(i, "NEW_DEF_RELATED_BUDGET",
                         ds.getItemNumber(i, "EFF_BUDGET") * ds.getItemNumber(i, "PCT_
APPL") * coef);
        ds.setItemNumber(i, "BUDGET_CHANGE",
                         newBudget - oldBudget);
        // Projected work quantity (inflation removed upstream so volume isn't
distorted)
        ds.setItemNumber(i, "WORK_AMOUNT",
                         Math.round(amount * ds.getItemNumber(i, "PCT_APPL") * coef));
    }
}
```

A few principles are worth drawing out for an engineering audience — each one a design choice the consultant specified, not a platform default:

- **A single coefficient carries the policy.** The ratio of the scenario budget to the current budget (`coef`) is computed once by the scripted model and then *propagated* through the decomposition. New crew budget = historical crew budget × `Percent Applicable × coef`. This keeps the math consistent from the statewide total down to a county crew.
- **Inflation is a first-class, separable term.** Inflation is folded into the cost coefficient so the *dollar* forecast reflects future-year prices. But because inflation must **not** distort the *physical quantity* of work required, the engine conditionally strips the inflation factor out when projecting work quantity (dividing by 1 + `inflation/100`). Cost and quantity share a model but are not allowed to contaminate each other — a subtle correctness requirement that a spreadsheet almost always gets wrong, and one the consultant caught precisely because of deep familiarity with the agency's intent.
- **Graceful data fallback.** Real historical data is sparse: a given district may have no recorded cost for an activity in a given year. The model is explicitly designed to **fall back to the computed statewide unit cost** when a district-specific unit cost is null. This single rule — proposed by the SME after observing where the agency's records had gaps — is what lets the engine produce a complete, defensible plan even where the historical record has holes, without silently producing zeros.

The result is an architecture where condition surveys, inventory quantities, and historical costs are continuously summarized in the background, and a planner's keystroke triggers a transparent, auditable chain — *survey score → LOS grade target grade → cost coefficient → multi-year budget → per-crew decomposition*. That chain is the product of design, not luck.

SECTION III

Streamlining the User Experience: From Desktop to Automation

A correct engine is necessary but not sufficient; planners will not adopt a tool that demands hundreds of manual entries. The refinements that made the engine usable did not surface from a feature roadmap — they came from the consultant sitting beside end-users, watching where they hesitated, where they re-keyed data, and where a multi-step chore could become a single click. Each friction point observed in the field became a backlog item the Product Owner championed and shepherded through successive sprints, working hand-in-hand with the core development team. The result was three right-click automations in the **Desired LOS Plan** window, each one a direct answer to an observed pain point.

1. **"Copy from Defect Survey" — bootstrapping the plan.** Rather than retyping the network's condition, the planner right-clicks and pulls the entire defect-survey summary for the chosen year into the plan. Multi-select (with standard CTRL/SHIFT semantics) lets them lift dozens of performance-measure/district rows at once; the system seeds each row's *Current LOS Score, Current Grade*, and a starting *Target LOS* automatically. (An early sprint defect — the picker popup failing to open — was diagnosed by the consultant against real client data and resolved with the development team so this entry point works reliably across all districts, including the all-districts login.)
2. **"Fill Scenario Costs" — the macro that populates a multi-year grid instantly.** This is the centerpiece, and the clearest example of consulting-driven ROI. End-users were editing target grades row by row and waiting on recalculation; the SME recognized that the value was in *bulk* what-if analysis and championed a one-command macro. With a single right-click, the engine calculates *Scenario Cost Year 1* and the projected *LOS Year 1–5* grades for **every** row in the plan at once — invoking the server-side cost model rather than looping in the browser. The same recalculation fires automatically whenever the planner edits a *Target LOS* grade or *Years to Target LOS*, so the grid behaves like a live financial model. Under the hood, the command marshals the plan rows to the server, runs the scripted model, and re-imports the computed columns — the heavy work never blocks the UI:

```
case "fill_desired_cost":
    if (sc_3.Execute(new CHTTPParam("command_id", "fill_desired_cost"),
                     new CHTTPParam("type_data", dw_type.ExportXML(dw_type.GetRow())),
                     new CHTTPParam("data", dw_1.ExportXML())) != -1) {
        dw_1.ImportXML(sc_3.ParserDataFromServer("data"));
        dw_1_RowFocusChanged(row_in);   // re-activate Save once the grid is populated
    }
    break;
```

3. **“Show Total Budget” — instant aggregate feedback.** Executives think in totals, not rows — a point the consultant heard repeatedly in stakeholder reviews. A right-click sums the scenario cost across the entire plan and surfaces it in a single formatted popup, turning a multi-hundred-row grid into one headline number on demand:

```
case “show_total_budget”:
    var total = 0;
    for (var i = 1; i <= dw_1.RowCount(); i++) {
        total += dw_1.GetItem(i, “EFF_BUDGET”);
    }
    basic_window.ShowAlert(1093, Utils.FormatNumber(total, ‘###,###’));
    break;
```

Beneath the selected row, a read-only **Activities pane** decomposes the chosen district/measure budget to the *Admin Unit* (crew) level, displaying the *Old vs. New Defect-Related Budget, the Budget Change,* and the projected *Scenario Year-1 Quantity* of work — each tagged with its `Percent Applicable` share so the lineage of every dollar is visible.

The net effect: a workflow that once meant maintaining a workbook per district now takes a handful of right-clicks, with multi-year scenarios populated in seconds and a defensible audit trail from grade to crew. None of these enhancements were inevitable — they were the product of an SME who treated user friction as a problem worth solving, year over year.

SECTION IV

Results and Strategic Outcomes

The strategic value of the framework shows up in two places: **visibility** and **justification** — and in both cases the platform supplied the mechanism while the consultant supplied the meaning.

Objective performance visibility — the District LOS Scorecard. Because every performance measure resolves to a letter grade on a consistent scale, the system can render a **LOS Report Card by District** — one page per district, organized by asset category (Bridge, Drainage, Pavement, Roadside, Traffic). The layout is deliberately glanceable:

- **Five color-coded grade columns** — A (blue), B (green), C (yellow), D (orange), F (red) — with each performance measure's grade placed in its column.
- **Intensity encodes the modifier** — within a column, the +/- variants shade from lighter to darker (A+ lighter than A, A- darker), so a reader sees not just the grade band but the trend within it.
- **Prior-year fallback with an asterisk** — if a district/measure has no sample for the selected survey year, the report back-fills the prior year's score and flags it with an asterisk and a footnote, so leadership never sees a misleading blank.

Illustrative excerpt of a single district's scorecard (anonymized, color described in brackets):

Category	Performance Measure	A	B	C	D	F
Bridge	Bridge Cleaning & Sweeping	A+				
Drainage	Culvert Cleaning	A				
Drainage	Storm Drain Erosion					F
Pavement	Asphalt Surface Defects			C-	D	
Pavement	Asphalt Rutting	A+				
Roadside	Mowing				D	
Traffic	Guardrail	A				

The scorecard's design was itself a consulting deliverable: the SME specified that it read its grading SQL from the *same* configuration the application uses, so it stays in lockstep with the live model — change the grading expression in the front end, and the report reflects it automatically — and that it back-fill missing samples rather than print blanks, a requirement that came directly from how leadership intended to use it. This converts an opaque pile of survey data into a board-ready artifact: a yearly and long-term picture of exactly how the network is performing, by region, in a language non-engineers can act on — and it exports cleanly to PDF for distribution to leadership and the public.

Data-driven budget justification. The deeper outcome is the ability to defend funding with evidence. Because the engine ties a *specific dollar amount to a specific, measurable grade improvement* — and decomposes it to the crews who will do the work — agencies can walk into a budget hearing and answer the only question that matters: *"What does this money buy, in condition the public can see?"*

> *For the authority profiled here, that capability underwrote a defensible **30%+ return on investment (ROI)** in maintenance outcomes.*

The significance is not the percentage itself but its provenance: the gain was justified by hard, network-wide condition data and a transparent cost-to-grade model, not by a narrative estimate. And that provenance was a joint achievement — a capable platform, yes, but one molded to the agency's reality by a consultant who understood both the software and the agency's operations well enough to make the numbers defensible. The same framework that justifies an expansion can equally justify a *reallocation* — steering dollars toward the districts and measures where each dollar buys the most grade improvement.

SECTION V

The Road Ahead

The migration from spreadsheet to dynamic LOS engine is best understood not as a tooling upgrade but as a shift in *operating model* — from periodic, manual reconciliation to a continuously-summarized, always-current decision layer. Three trajectories follow naturally:

- **From descriptive to predictive.** With multi-year LOS projections (Years 1–5) already in the model, the next step is optimization: letting the engine recommend the target-grade and spending mix that maximizes network condition under a fixed budget ceiling, rather than asking the planner to discover it by trial.
- **From periodic surveys to continuous sensing.** As field data collection moves toward mobile GIS capture, telematics, and imagery-based condition assessment, the same summarization architecture can ingest higher-frequency data without changing the analysis layer — the separation of aggregation from analysis is what makes this future-proof.
- **From configuration to true self-service.** Because performance measures, grade scales, and cost models are already *data and pluggable scripts* rather than hard-coded logic, agencies can evolve their own standards without vendor release cycles — the platform becomes a framework, not a fixed product.

There is a broader lesson here for any agency staring down its own spreadsheet chaos. Powerful asset-management software is necessary, but on its own it delivers only a fraction of the outcome — perhaps a fifth. The rest is translation: mapping legacy processes, encoding localized economics, owning a backlog of targeted enhancements, and standing as the bridge between what the business needs and what the platform can be made to do. In this engagement that translation was the work of a **Product Owner** — and it is precisely that work, repeatable across agencies, that turns a capable platform into measurable value: an application and workflow set capable of delivering a 30%+ return on investment.

The encouraging implication is that these outcomes are not unique to one authority. Any DOT facing the same combinatorial spreadsheet burden can reach the same place — a standardized MQA program, a dynamic LOS engine, board-ready scorecards, and evidence-based budgets — by pairing the platform with an experienced Product Owner who knows how to mold it to a specific operational reality. When condition is standardized into

*That is the real transition underway across the public-asset sector: not from one tool to another, but from **managing history to engineering the future — with the right people at the wheel.***

Prepared as a generalized, anonymized account of a statewide DOT MQA modernization led by a Product Owner. Figures reflect a past, legacy customer engagement and are shared illustratively; no current client-specific names, geographies, credentials, or proprietary scripts are disclosed.

About the Author

Yash Pamulapati
Product Owner / Consultant (Professional Services)
AECO, Trimble Inc.

Learn more at assetlifecycle.trimble.com,
or request a Demo of Trimble Unity Construct

LEARN MORE

REQUEST A DEMO

Trimble