Sokil Logistics

A Case Study

KIM SOKIL
DIRECTOR, INFORMATION TECHNOLOGY

Industry: Logistics

Employees: 200

Incorporated: 1951

Locations: 3

ParaSQL Applications In Production: 16

Size of Development Team: 1

Integration Use Cases: Freight Forwarding Partner API Integration for Ladings & Deliveries

Mobile Use Cases: Signature Capture for Electronic Proof of Delivery on Tablets in Trucks

Background

The Sokil Group consists of six companies that provide transportation and logistics across Canada. It is a family run company first incorporated in 1951. They have three primary locations across Canada, 200 employees, and almost as many trucks. They haul any type of cargo except liquids, such as petroleum.

Kim Sokil, a granddaughter of founder John Sokil, is the third generation of family to manage the firm. Kim has spent her entire career in IT. In the early part of her career she worked at Sokil for a few stints, but then went off to a career that included Telecom, Medical systems, Government Agencies and IT consulting. Over the years she had implemented many ERP and other packaged applications. She returned to Sokil in 2015 only to find a system that she had helped implement in 1992 was still alive and kicking, but arguably past end of life. That platform was an IBM AS/400 system originally based on a package but highly modified. In the intervening 20+ years the hardware had been upgraded once, but IBM had informed them that it would no longer be supported.

The Evaluation Process

Kim made one of her first priorities to move functionality from this system to the next system. One of her first decisions, that is shared a lot these days, was to get out of the business of managing computer hardware and move the system to the cloud. It was her second decision that is the focus of this case study.

Kim decided she was not buying a package. This is still a minority decision, especially for a small to medium business. I asked her why. She told me in her consulting career she had worked on a great many package implementations. In her experience, most didn't deliver the functionality that the client needed. They rarely completed on time and were poor value for money. The base packages that most firms started with were rarely even close to the functionality needed. Further, she said once you commit to a package the vendor will provide upgrades which are often disruptive, and if in the inevitable case that you have modified the package, no longer forward compatible. She knew enough to know what she didn't want.

She set out on a quest to find a platform or an environment or a set of tools where she could build and maintain the functionality they needed.

Over the next 6 months she engaged in over a dozen trial implementations. She would bring a product in house, build a representative part of a solution and explore how easy was it to build screens, reports, validation and the like. Most were not up to her expectations.

Then she tried a product called ParaSQL from a company of the same name, which is bicoastally located in Florida and Washington State. The ParaSQL system is a model driven, no code / low code environment, and while that wasn't what Kim was looking for specifically, she recognized its capability when she tried it out.

Applications Developed

She started by building out a manifesting system and a fuel management system, as these were a couple of systems that were in the most need of replacement from the previous application suite. This went well, and went into production, which provided a great deal of confidence. In early 2016 she committed to a path of full conversion by spring of 2019 when IBM said they would no longer support that model of AS/400. She is well on track having built the following applications as of late 2018:

- Billing – a robust application that connects invoicing to quotes for ease of billing
- Freight Integration – API integration application that connect two servers together – IoT
- Sales – to capture all sales activity
- Delivery – an application used by the drivers to mark deliveries completed and get electronic signatures (similar to UPS)
- Probills and Manifests – used by transportation industry to move goods, also sending customer emails of invoices and related documents.
- Warehousing – inventory capture information
- Reports – all management reporting
- Customer & Quote – tracking customer information and related quotes
- HR & Payroll – simple information capture for resources, used to feed vendor payroll systems
- Fleet – capturing all fleet information
- Customer – embedded in Sokil website so customers can self-serve. See delivery information, retrieve POD documents, invoices and statements.
- Telematics – captures all the data recorded from GeoTab relays – engine data, trip data, driver safety data
- Fuel – since fuel is a large part of a transportation expense special reporting and data capture are needed for reporting taxes
- Courier – they deliver small packages
- A/R & A/P – simple document capture
- Content Management – they store all their scanned documents in the same database, indexed to the entities they correspond to.

Although she hadn't heard the term "Data-Centric", this is exactly what this is. She said she had always had a philosophy of "data first" and was surprised that younger generations of application developers did not. Sokil's systems run on a single simple data model consisting of fewer than 50 tables. There is a very small amount of code (a few exception routines, triggers and notifications need to be written in PL/SQL or JavaScript, but these are the exceptions). She is very intentionally avoiding writing code as much as possible, partly for maintainability and partly to help those to whom she will eventually turn over the care and feeding of the system.

Maintainability

The system has all the attributes that are far too often missing in deployments of this type, including ease of understanding and ease of change. I asked about the effort to make a routine change and a difficult change. By routine change we agreed that something like adding a field to the database, getting that on a form and providing formatting and validation would be a routine change. An example of a larger change would be something like adding a fuel surcharge feature, which might affect several modules, as it would have to be set up, maintained and carried all the way through invoice calculation and reporting.

She said the routine changes typically take an hour or two, and she can slip stream them into production. She has enough experience with the system that she knows which changes will not be disruptive. The platform is built such that these changes do not require data conversions, or even shutting the system down, a user of the system will get the change in the next refresh. The larger changes she said typically take 2-3 days.

I asked her to compare that with her experience with packaged implementations, and she laughed. Even the simplest of changes, say to change the heading on a report, starts a work flow of definition, quoting, scheduling, development, QA, etc., etc., each of which typically takes weeks or more to get from requirements to in production.

Team Size

All this functionality was built by Kim herself over the course of a few years. Most would agree that traditional development would have required a medium sized team on each of the more than a dozen modules she has implemented over nearly that same time frame.

This study shows that even medium sized companies can benefit and benefit significantly from this approach. It shows that the tools are real and the approach proven. And it shows that the data-centric approach delivers the promised value, even if you weren't aware that is what you set out to do.

Customer's Summary Evaluation

Pros: Fast development, great built-in features, not a lot of coding needed, supports all of our ERP needs.

Easy report generation. Knowledgeable support resources. Allowances for larger applications.

Quick publishing, can be done while users are on the system - thus no down time. User security roles down to the field/column level. Group security available. Living in Google land allows our users to adapt to in-house applications quickly as the look and feel is very familiar. No hardware, networking needed. Pay for what you need, grow when you need it. API integration.

Cons: At this time we do not have any - with all the new features and functions released every few months so far we are able to do what we need - and more!

Overall: Ability to develop through prototyping allows us to rapidly have new versions of software releases done within days and weeks. Over the last 3 years we have been able to grow when needed and finance appreciates a monthly fee. Leaping into new software features for us has given us a competitive advantage over much larger companies.