QOS SOFTWARE Development

SCOPING YOUR SYSTEM INTEGRATION:

AN HONEST PRE-PROJECT CHECKLIST

877.800.7672

qossoftware.com

Scoping Your System Integration: An Honest Pre-Project Checklist

The most expensive integration decisions get made before anyone writes a line of code — in the gap between “our systems don’t talk to each other” and a clear picture of what must connect, which way the data moves, and who owns the result. This checklist closes that gap.

Work through it before you commission anything. The clearer your answers, the tighter your scope, and the fewer surprises later — and the easier it is for any honest development shop, ours included, to tell you what to build, what to buy off the shelf, and what to skip. There are no prices in here on purpose: scope comes first, and cost follows scope.

1. Start With the Problem, Not the Connectors

It’s tempting to open with “connect tool A to tool B.” Start with the problem instead — the connection you actually need falls out of it, and half the syncs you imagined turn out to be unnecessary.

Answer First:

- What breaks today because two systems don’t share data?
- What does someone re-key by hand, and how often?
- What decision or report is late or wrong because the numbers live in more than one place?
- What happens to the business if you do nothing?

The Rule:

If you can’t name the manual work the integration removes, you’re not ready to scope it yet. Get it down to one sentence first.

2. Inventory Every System That Must Connect

Almost no integration is just two systems. List every application that touches the data before you design anything — the ones you forget are where projects overrun.

List Your Systems:

- Which applications hold the data — CRM, accounting, ERP, e-commerce, a database, a spreadsheet?
- Does each one have a modern API, or is it legacy or on-premise?
- Is it cloud-hosted, self-hosted, or a mix?
- Who administers each system, and who can grant access?
- Is any system due to be replaced soon — and would that waste the connector?

Why It Matters:

Every system on the list adds design, testing, and a failure case. A clear inventory turns an open-ended risk into a plannable one.

3. Map the Data: What Moves, and Which Way

An integration isn't "linking" two tools; it's a specific set of data moving in a specific direction. Trace each flow before you commit to a design.

Trace Each Flow:

- What exact records move — contacts, orders, invoices, inventory?
- Which system is the source of truth for each field?
- Does data flow one way, or in both directions?
- What happens on a conflict — which side wins?
- Does anything need transforming or reformatting on the way across?

The Rule:

Name one source of truth per field. Two-way sync without a clear winner is how duplicate and overwritten records begin.

4. Pin Down Volume, Frequency, and Freshness

The same two systems can be a small job or a large one depending entirely on how much data moves and how quickly it must arrive. Decide the numbers — in scale, not exact figures — up front.

Decide Early:

- Roughly how many records move per day, and in what bursts?
- How fresh must the data be — real-time, every few minutes, hourly, or nightly?
- Is there a historical backfill to load once at the start?
- Do any systems have rate limits or quotas that cap how fast you can sync?
- Will volume grow, and how much headroom should the design carry?

Why It Matters:

Freshness and volume are where a simple connector stops being enough. Real-time, high-volume, or rate-limited flows are exactly where custom work earns its cost.

5. Settle Authentication, Access, and Security

Every connection is a door into a system that holds your data. How that door is secured is a scoping question, not an afterthought — settle it before the build.

Get These Right:

- How does each system authenticate — API key, OAuth, a service account?
- Who owns the credentials, and where are they stored safely?
- What is the least access each connection actually needs?
- Does any data carry compliance or privacy requirements in transit?
- How are credentials rotated, and what happens when one expires?

The Bottom Line:

Insist on a documented credentials map you own. An integration whose access no one can account for is a security gap waiting to be found.

6. Plan for Failure: Errors, Retries, and Monitoring

Integrations fail quietly — an API changes, a system is down for an hour, a record is malformed. What happens next is the difference between a blip and a silent data problem you find weeks later.

Plan For:

- What happens when a transfer fails — retry, queue, or alert someone?
- Are errors logged somewhere a human will actually see them?
- How do you know the integration is running, not just assumed to be?
- What happens when a connected system changes or versions its API?
- Who is responsible when something breaks, and how fast must it be fixed?

Why It Matters:

An off-the-shelf connector gives you whatever error handling it ships with. When a silent failure would cause real harm, catching, logging, and retrying on your own rules is worth building.

7. The “Is an Off-the-Shelf Tool Enough?” Test

Plenty of integrations should be built on Zapier, Make, or Power Automate — and we’ll tell you when yours is one of them. Run this test before you commission anything custom.

Off-the-Shelf Is Probably Enough When:

- It’s one simple trigger and one simple action.
- Both tools already offer solid, supported native connectors.
- Volume is low and nothing critical breaks on a missed sync.
- The logic doesn’t branch or transform much along the way.

Custom Earns Its Cost When:

- The logic is multi-step or a connector simply can’t express it.
- No connector exists for a niche or legacy system.
- Per-task pricing gets expensive at your volume.
- Errors must be caught, logged, and retried reliably.

8. Decide Ownership and Life After Launch

An integration is never “done” — systems it depends on keep changing. Settle who owns it and who keeps it healthy before the first line of code.

Get These In Writing:

- Who owns the source code and the credentials map when it’s done?
- Can you change developers later without rebuilding from scratch?
- Is there a per-connector or per-task fee to keep it running?
- Who maintains it when a connected system updates its API?
- How are new flows requested, prioritized, and added?

The Bottom Line:

Software you don’t own is a subscription you paid to create. Insist on owning the code, the documentation, and the credentials — in the contract, before work starts.

The Final Word

An integration is only as good as the thinking that goes in before the build. Answer these questions honestly and you’ll walk into any conversation with a development shop knowing exactly what you need — and able to tell whether they’re solving your problem or selling you a project.

Next Step:

Fill in your answers to sections 1, 3, and 8 first — the problem, the data flow, and ownership. Those three decide most of the scope. When you’re ready to turn the checklist into a plan, QOS Software will scope it with you honestly, and tell you where an off-the-shelf connector would serve you better. Call **877.800.7672** or visit **qossoftware.com**.