Puzzl Media

ARTICLE

How to Budget for Custom Software: What to Know Before Getting a Quote

by Nooruddin Hargey | 2 October 2026

Product Strategy | Software Development

Understanding what actually drives the price, before you request a number

Ask three development companies to quote the same project and you'll often get three numbers that don't just differ, they don't agree on what's being built. One vendor scoped the login flow you mentioned in passing. Another assumed you'd handle it with a third-party auth provider and didn't price it at all. A third read "admin dashboard" as a full permissions system with audit logging, because a past client burned them for not asking.

None of them are lying to you. They're answering slightly different questions, based on slightly different assumptions, because the brief left room for it. That's the actual reason custom software quotes vary so much, and it's not a reason to distrust the process. It's a reason to understand it before you're in it.

This isn't a piece about what custom software costs. Anyone who gives you a confident number without knowing your requirements is guessing, and anyone who gives you a range wide enough to be honest isn't giving you anything you can budget against. What's actually useful is understanding what drives that number up or down, what a vendor needs from you to narrow it, and how to tell a good quote from a bad one once you have a few in hand.

Why This Is Genuinely Hard to Budget For

There's a well-documented reason early software estimates are unreliable, and it isn't vendor incompetence. Software engineer Steve McConnell popularised the idea of the "cone of uncertainty": at the very start of a project, before requirements are pinned down, estimates can realistically be off by a factor of four in either direction. As the requirements get more specific, that error margin narrows, and by the time you're deep into build, the estimate is close to the actual cost [1].

That's not a flaw in how developers estimate. It's a description of how much genuinely isn't known yet at the point most businesses ask for a number. You haven't fully decided what "user roles" means for your business. You don't yet know whether that third integration has a documented API or a support ticket queue. The vendor doesn't either, until someone looks.

The practical implication: a quote you get in a first conversation is not the same kind of number as a quote you get after a proper scoping exercise. Both are useful. They answer different questions, and it's worth knowing which one you're holding.

What Actually Determines the Cost

Underneath every custom software quote, the same handful of variables are doing almost all the work. None of them map to a fixed price per feature. A "login system" can be a day's work or three weeks', depending on everything below. But understanding them lets you read a quote instead of just receiving one.

Scope and feature count

The obvious one, but scope is rarely as fixed as it sounds in a first conversation. "A booking system" could mean a single calendar with no conflicts to resolve, or a multi-resource, multi-timezone system with cancellation policies, waitlists and refund logic. The vendor needs to know which one you mean.

Complexity of the underlying logic, independent of feature count

Two features can look identical on a spec sheet and take wildly different effort, because one has three straightforward rules and the other has eleven edge cases that only surface once you sit down to build it. A discount system with one flat percentage is simple. A discount system with stacking rules, exclusions and expiry dates is a different piece of software wearing the same name.

Integrations

Every external system you connect to, whether it's a payment processor, an accounting package, a CRM or a legacy internal database, brings its own documentation quality, its own failure modes, and its own testing burden. A well-documented, modern API might take a day to integrate cleanly. A legacy system with patchy documentation can take weeks, and you often can't tell which one you're dealing with until someone actually opens the docs.

User roles and permissions

"Admins can do more than regular users" sounds simple until you enumerate what "more" means across every screen and action in the system, and whether permissions need to be role-based, resource-based, or both.

UX and design work

A functional-but-plain internal tool and a polished customer-facing product take fundamentally different amounts of design effort, even if the underlying features are identical. This is often where budgets get set unrealistically low, because design doesn't show up as a "feature" the way a login screen does.

Platforms

Web-only, mobile-only, and web-plus-mobile aren't the same project scaled up. They're different projects that happen to share business logic. Supporting both a website and native apps roughly doubles the surface area that needs building, testing and maintaining, even when the underlying data and rules are shared.

Non-functional requirements

How many concurrent users does this need to handle? What happens if it's down for an hour? Does data need to sync in real time, or is a five-minute delay fine? These questions rarely appear on a feature list, but they quietly determine the architecture, and architecture decisions are expensive to unwind later.

Security and compliance

In South Africa, if the system touches personal information (customer records, health data, payment details), it's subject to the Protection of Personal Information Act (POPIA). The act requires lawful processing, defined purpose, adequate security safeguards and, for many organisations, a registered information officer. Non-compliance carries fines of up to R10 million and, in serious cases, imprisonment [2]. Building those safeguards in from the start is different work, and different cost, than bolting them on after the fact. If the business also serves EU customers, GDPR brings its own, broadly similar obligations on top.

Team composition and rates

A quote also reflects who's doing the work, seniority mix, whether the vendor is a solo freelancer, a boutique studio or a large agency, and where they're based. This is a real driver of quote variance and worth naming honestly. It doesn't make a higher number better or a lower one worse on its own. It's one more thing to ask about, not assume.

A Concrete Contrast

Two examples make the scope-driven nature of cost easier to see than a list of variables does.

Take an internal tool for a 12-person operations team: a system to track jobs from intake to completion, with one user role, no payments, no public-facing interface, and a single integration into an existing spreadsheet or database. That's a genuinely contained build. The logic is knowable in advance, the integration surface is small, and there's no external attack surface to defend beyond normal good practice.

A customer-facing platform with payments, multiple user roles, several third-party integrations, a mobile app alongside the website, and workflows that vary by customer type is not a bigger version of the same project. It's a different category of project. Each integration is its own mini-project of discovery, error handling and testing. Payments bring PCI-adjacent (the card industry's data security standard, PCI DSS) obligations even when a processor like Stripe or Paystack handles the sensitive card data directly. Multiple roles multiply the number of paths through the system that need to be designed and tested. None of this means the second project is a bad idea. Plenty of businesses need exactly that kind of system. But going in expecting it to cost roughly what the first one did is where budgets fail before the project starts.

What to Have Ready Before You Ask for a Quote

You don't need a finished specification to get a useful quote. You do need to have thought through enough that a vendor isn't guessing on your behalf. Before that first conversation, it's worth having a real answer, even a rough one, to each of these:

- What is this software actually for, and who uses it? Internal staff, customers, both?
- What are the two or three things it absolutely must do on day one, versus what's a "nice to have" for later?
- What existing systems does it need to talk to, and do you know if those systems have a documented API?
- Does it need to handle payments, personal information, or anything else with a compliance dimension?
- Web, mobile, or both, and if mobile, does it need to work offline?
- Roughly how many people will use it, and does that number change seasonally or with growth?
- Do you have existing data that needs to migrate in, and what state is it in?
- What's your actual timeline, and is there a hard external deadline driving it (a regulatory date, an event, a contract) or is it more flexible than it feels?
- Who internally will be available to answer questions, review work, and test as it's built? This isn't a vendor-side cost, but it's a real cost to the project's timeline and quality if nobody has the bandwidth.

A vendor worth working with will ask most of these anyway. Arriving with answers doesn't just speed things up. It's the single biggest thing you can do to narrow McConnell's cone before anyone writes a number down.

Why Vendors Quote Differently for "the Same" Project

Assuming everyone read your brief in good faith, quote variance usually comes from a handful of predictable places.

Different assumptions filling the same gaps

If your brief doesn't specify whether "reporting" means three fixed charts or a flexible report builder, two vendors will fill that gap differently, and their numbers will reflect two different products.

Different amounts of built-in risk buffer

A team that's been burned by scope creep on similar projects will price in more contingency. That's not padding for its own sake. It's a rational response to how often "simple" projects turn out not to be, and Boehm and Basili's widely cited research on rework found that fixing a problem discovered late in a project can cost tens of times more than catching it during requirements and design, with the ratio far smaller on small, well-scoped projects than on large ones [3]. A vendor who's seen that curve firsthand prices accordingly.

Different definitions of "done."

Does the quote include testing? A staging environment? A short warranty period after launch where bugs get fixed for free? Documentation? Deployment support? Two quotes with the same headline number can include meaningfully different amounts of actual delivered work.

Different team structures

A quote built around one experienced generalist looks different from one built around a small team with a designer, a backend developer and a QA person. Neither is automatically right (it depends on the project), but it's worth understanding which one you're being quoted.

Different pricing models

Fixed-price quotes bake risk into the number upfront. Time-and-materials quotes shift that risk onto you but usually come in lower on paper. Neither is a trick, they're different ways of allocating who absorbs the cost of the unknowns that the cone of uncertainty guarantees exist.

The Development Quote Is Not the Total Cost

This is the distinction that causes the most budget pain after the fact, and it's worth being direct about it: the number in your development quote is the cost to build the first version. It is not the cost of owning the software.

Decades of research into software lifecycle costs converge on a consistent, uncomfortable pattern: across a system's working life, maintaining and evolving it typically costs more than building it did in the first place, often two to four times more once you total everything from bug fixes and security patches to dependency upgrades and ongoing feature work [4]. None of this is a reason to avoid custom software. It's a reason to

budget for it as an ongoing responsibility rather than a one-off purchase. The shortcuts that quietly build up into technical debt are often exactly what turns a reasonable development cost into an unreasonable maintenance bill down the line.

Total cost of ownership includes, at minimum:

- **Development:** the build itself, which is what most quotes actually price.
- **Infrastructure and hosting:** servers, storage, bandwidth, usually billed monthly and scaling with usage.
- **Third-party services:** payment processing fees, SMS or email delivery, mapping APIs, any service you're paying per-use or per-seat.
- **Maintenance and support:** keeping dependencies updated, patching security issues, fixing the inevitable bugs that surface once real users start using the thing in ways nobody quite anticipated.
- **Future changes:** the business will change, and software built for today's business needs updates to stay useful. This is normal, not a sign the first build was done badly.
- **Security and compliance obligations:** not a one-off setup cost if the software handles personal information, POPIA's security safeguard conditions are an ongoing responsibility, not a checkbox.
- **Internal costs:** the time your own team spends reviewing work, testing, providing feedback and managing the relationship. Real, and easy to leave out of a budget entirely.

None of this means development quotes are misleading. It means a development quote answers "what does it cost to build this," and a business decision needs the answer to "what does it cost to have this," which is a different and larger question.

How to Compare Quotes Without Just Picking the Lowest Number

Once you have two or three quotes in hand, the comparison that actually matters isn't the total at the bottom. It's what's underneath it.

Start by checking whether the quotes are actually pricing the same thing. Ask each vendor to walk you through what's included, and just as importantly, what isn't. A quote that's silent on testing, deployment, or a post-launch support window isn't necessarily hiding something, but you need to know before you compare it to one that spells those out.

Ask how they'd handle a change mid-project. Not whether they will, because they will, but what the process looks like when it happens. A vendor with a clear, reasonable change-request process is telling you something useful about how the rest of the project will run.

Ask who actually does the work. A quote built around senior people costs more per hour and often less overall, because fewer hours are lost to rework and misunderstanding. That's not a rule, but it's worth asking rather than assuming.

Ask what happens after launch. Does support end the day the software ships, or is there a warranty period? What does ongoing maintenance cost once that period ends, and is it optional or effectively mandatory to keep the system secure?

And ask who owns what's built. Source code, design assets and any custom infrastructure should generally belong to you at the end of the engagement. That's a completely normal thing to confirm upfront and a very awkward thing to discover the answer to later.

None of this is about catching a vendor doing something wrong. It's about making sure the number you're comparing is actually comparable.

Questions Worth Asking a Development Partner

A short, direct list, worth taking into any first conversation:

- How would you scope this project before committing to a fixed number?
- What's included in this quote, and what's explicitly not?
- What's your process when requirements change partway through?
- Who, specifically, will be working on this, and what's their experience with similar projects?
- What does support and maintenance look like after launch, and what does it cost?
- Who owns the code, the design files and any infrastructure you set up?
- What's the realistic range this could land in, and what would move it up or down?

- Have you built anything with a similar integration or compliance requirement before?

A good technical partner won't be defensive about any of these. Their answers, more than their number, are what actually tells you whether they understand your project.

One thing worth saying plainly before you request that first quote: budget for what happens after launch, not just for getting there. A number that only covers the build is answering half the question, and the other half, the part covered above under total cost of ownership, is where most of the unpleasant surprises actually live.

You don't need certainty on all of it. You need enough to have a real conversation, rather than a guessing exercise dressed up as one. And if you haven't yet worked out whether building custom software is the right call in the first place, rather than buying or extending something off the shelf, that's a decision worth settling on its own terms before you start collecting quotes.

The honest version of this whole topic is simple, if not especially satisfying: you don't need to know exactly what your software will cost before you talk to a development partner. You need to understand what determines that cost, what information changes the estimate, and what you're actually comparing when three vendors hand you three different numbers. Get that right, and the number stops being a guess you're hoping lands close, and starts being something you can actually plan around.

If you're at the stage of trying to scope a project honestly before it turns into a sales conversation, that's a conversation worth having early. Talk to us.

Sources

1 Steve McConnell, *Software Estimation: Demystifying the Black Art* (2006); the "cone of uncertainty" concept was first named in his earlier *Software Project Survival Guide*. Supports the claim that early-stage software estimates carry a wide, well-documented margin of error that narrows as requirements firm up.

2 Protection of Personal Information Act (POPIA) in South Africa, summarised via multiple compliance-law overviews (Termly, Scytale, Secure Privacy). Supports the description of POPIA's scope, security safeguard conditions, and maximum penalties (R10 million fine, up to 10 years' imprisonment for certain offences).

3 Barry Boehm, *Software Engineering Economics* (1981); Boehm & Basili, "Software Defect Reduction Top 10 List," IEEE Computer 34(1), 2001. Supports the point that late-stage changes and defects cost substantially more than the same issue caught early, with the ratio meaningfully smaller on small projects than large ones. Cited here directionally, not as a precise multiplier.

4 Multiple independent studies converge on maintenance representing the majority of software lifecycle cost, including Schach (1999, ~67%), Pigoski (2001, >80%), and the IEEE Computer Society's commonly cited 60-80% range; O'Reilly's "60/60 rule" is frequently referenced as a rough industry heuristic. Supports the claim that owning software typically costs more than building it, over its working life.

ABOUT PUZZL MEDIA

Puzzl Media is a developer-led software studio in Cape Town, founded in 2021, working with clients in South Africa and abroad.

Who we work with: founder-led businesses building their first product, and established companies whose engineering or product teams need extra, specialist hands.

How we work: embedded alongside your engineers, or owning a build end to end, from discovery to support after launch.

Get in touch

hello@puzzl.co.za | puzzl.co.za
Puzzl Media Pty Ltd, 24A Hastings Rd, Muizenberg, Cape Town 7945, South Africa