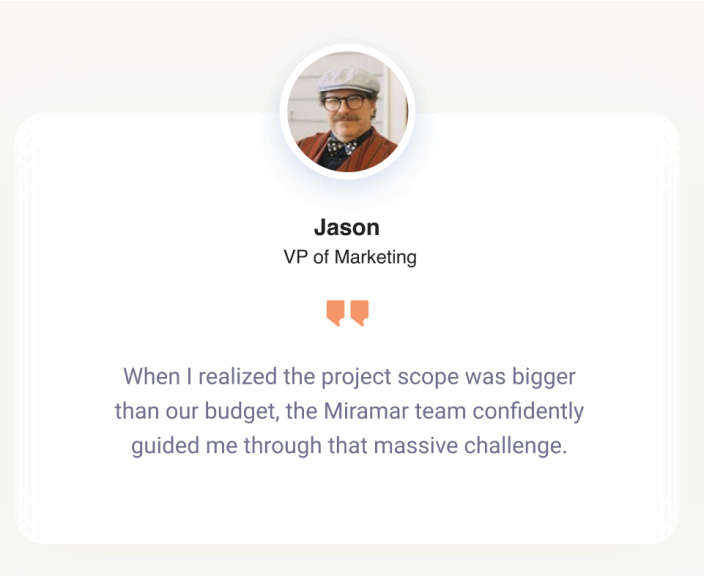
Budgeting for Software:
a 7-Step Guide

There are two types of software budgets. You can either **gamble** or you can **sprint**. The goal of this guide is to advocate for pace and predictability, over guesswork and hope, so businesses can continuously plan for more digitalization. Sprints are a one or two week cycle of work that are designed to prioritize, focus, deploy some working features, and repeat.

Knowing that gambling isn't really good enough for business planning, I am going to outline some of the most important things that experts have learned over the last two decades of trying to be on time and under budget.

Before you get started building, decision-makers will want to know how much your project is going to cost, and when it can be completed. I recommend you bring in 2 or 3 senior developers and describe to them three things:



1. The ultimate result you want from the software (i.e. "I want to make $50/year from existing

customers buying more through these added features")

2. The feeling you hope the User Interface communicates to your users (i.e. "I want it to be helpful and communicative like Turbo Tax")

3. The database constraints this project will need to integrate with or rely on (i.e. "We need to use our Hubspot marketing data to populate the customer's birthday")

Have each developer think about it for one day and then reconvene with a "Confidence Percentage" like a weather forecast. It would be something like this:

CONFIDENCE PERCENTAGE	40%	70%	90%
COMPLETION DATE	2 months	6 months	12 months
IMPLIED LABOR COST	$40,000	$120,000	$240,000

Like the weather, you should expect wild ranges of possibilities here, after all, you are going to build something that hasn't been built before - and your developers are going to have to adapt to the changing requests of the business stakeholders too. There will be bugs, there will be mistakes, and there will be ambiguous challenges to problem-solve. Your developers are critical thinkers and they are investigators. Software developers are not estimating how long it takes to make a ten-thousand-brick wall. There are going to be many unknowns.

Next, its time to get started with some intentionality. Use these next 7 things to frame your thinking and leadership.

3 Common Pitfalls

1. **Avoid predicting the future.** Casting vision is great, but you need to walk the balance between knowing that a software project will be worthwhile to tackle, but not trying to predict every feature it will have the date you need to go live, and the budget you need to stay under. This proves very hard, as most decision makers get excited about the raw potential of their new concept coming to life in software.

2. **Avoid overconstraining your team** by picking the order of priority between Budget, Completion Date, and Scope. If you ask for all three to be equally important, you're in for a rocky ride. In our company we prefer to think in terms of Completion Date, then budget, then scope. We know the easiest thing to change is our idea (Scope), and by publishing a smaller scope faster, we can get user validation (and some income!) before engineering more features.

3. **Avoid the "one-time-build"** mentality. Many people who haven't lead software projects before think they can get an estimate and then deliver their vision within a set budget request from the money people, and a reasonable timeframe. *But its more like buying a ranch property.* Yeah, you may get a sweet house on some acreage with tons of features - but with you'll find you are no different than all those other homesteaders: the weeds will need plucking, the fence will need new boards, and your utility bills will

never go away. Make sure you understand your software will need fresh paint every few years, the termites (bugs) will need a plan of attack, and you'll always need to update the kitchens and bathrooms to keep with the times. There is no such thing as one and done in software or home-ownership. 20% is a good number to set aside for annual maintenance, and then you can hope that it's lower than that.

4 things to budget for

4. **Build Strong Story Writing Expectations (5% of labor?) -** Set aside 8-10 hours a week of your Product Owner's time to write out some scope in greater detail than they usually want to. They need to stay WAY ahead of the development team. These hours are just for thinking and writing, not even for meetings around the project. It's hard to do this part right if you've never done it before. The larger the spend on developers, the larger you need the time investment of the Product Owner (and project manager for that matter) to be. If they are new to the Product Owner game, make sure they brush up on **how to write a user story that devs can actually use.** It's too easy for the Product Owner (or often the "subject matter expert") to rely heavily on meetings for communicate how they want a feature to work and act. Developers don't thrive in a meetings culture - they need all the edge cases of a feature explained in writing, or they will certainly forget things when they're knee deep in code and have been critical thinking for 6 hours straight. If you don't write clearly and thoroughly, you'll rack up an incredible amount of what we refer to as **technical debt**. Don't forget the "Acceptance Criteria" details

and don't be lazy.

5. **Real Design Pros (5% of labor?)** Front End Developers *are not designers*, but they often are used in this way, and apps become more cumbersome, look worse, feel worse to your users, and **cost more to develop** because of the common trappings of ambiguity. If you had an engineer draft your new house drawings, you are going to get something that functions, but will not necessarily flow. My wife would kill me if I let the engineers choose how the kitchen and living room to tie in together. Design by real software designers will make decision-makers feel better about features, so they will approve them faster. Designers will make developers feel clarity when designs are combined with the Product Owner's story-documentation, so they will develop faster. Product Owners will write scope faster because they can *see* what they need to *document*. Everyone will go faster, you will spend less money because of it, trust me, we've done this a few times.

6. **Automated Tests & Documentation (10% of labor?)** Documentation is skipped on 50-60% of projects because the projects are simply underfunded. If you do that, you will regret it in a year or two, guaranteed. Maintenance costs will go up when your current developers will turn over (the average developer stays around for 2 years in the USA). New developers will struggle to understand the reasoning behind the original code pattern choices, or what precedents are set. They need documentation to tell them *why and how.* "What" isn't good enough. Your code will become like a tangled ball of yarn. My customers have spent millions of dollars on software development labor to redevelop successful applications because they are no longer sustainable to keep up with. The other side of the same coin - is writing

automated tests. This is a practice that top-notch developers would never skip. They know its too important. Automated tests are written right after a feature is developed, and protect you from pesky bugs that you spend countless hours of labor to troubleshoot. Read more about **automated tests** and why you need to spend money on them. Please trust me. If your developers don't write automated tests naturally, they likely are a junior developer or have grown up through the ranks of dysfunctional teams.

7. **Stakeholder Weekly Meetings (3 hours a week)** - Don't skimp on the meetings that will manage a project into a success. The miscommunications and "oh shit" moments that come from getting out of touch with the momentum of your project can really cost you. **There are a thousand small decisions to make, and they need to be made on time - which is ahead of time**. I recommend two weekly 90 minute meetings for all involved (lead by your Product Owner). Review your backlog, and get continuous estimates. Don't micromanage, and don't shame during these meetings - the project is going to take as long as its going to take. Instead, support and **cheer for momentum**.

Building Software for the Lowest Cost Possible is all about Momentum.

In closing, a word on software-team momentum. At Miramar, we use point-estimates (opposed to hour-estimates) to free our developers from the stress of time. Points are added for complexity, ambiguity, precedent, and more (we like to use Fibonacci numbers). Once we estimate and build, estimate and build, and repeat for a month or so, we

start to develop a history of how many points we can get done on that project, with this team, in a given week. We start to look for ways to get a few more points here and there. We start to see how many bugs will naturally arise on this software - and we start to be able to predict how much we can get done.

By doing this type of estimating, we can have confidence that we trend upward on our production, over time. The first couple of months of a big project are always the slowest. Backend architecture goes into place, code design patterns are chosen, tech stack becomes more familiar, and trust between the team members is gained.

The lowest cost of software is always after the initial months, assuming leaders are supporting and not overburdening the team. If leadership instead rushes urgent delivery dates or interrupts the developer's flow with every imperfect bug - there will never be real momentum gained, and the project will simply be more expensive than it could be.

Be a good leader, and ask your team how they can get more momentum, and what you can do to support that.

TLDR:

- Take the developer's estimates and use "confidence percentages".
- Add 30% to their estimates for design, QA, documentation, automated tests, and product stakeholder meetings.
- Make sure you are building the right thing, bring in **a consultant** to validate your plan if needed.
- Gain momentum through present and supportive leadership.
- Plan on 20% recurring annual maintenance for a successful product. You will, after all, want to repaint your house.

- Don’t spend your whole budget - save money for digital marketing and user training too!

Still Reading? Miramar can help you!

If you just need more hands-on-deck look at our On-Demand Service to contract a needed expert. You can trust the staff we offer, they are battle-tested and were selected for top communication skills.

If you need someone to just handle it all for you, our studio team is set up to provide every single software expertise needed to deliver production-quality software to our clients. We build them and we can maintain them - or we can take over the project or website that got too big for your last freelancer.

Miramar works with fortune 100 companies or innovative SMB leaders. We are trusted by our clients, and we do this stuff with excellence.