Why Software Projects Fail

(and How to Make Yours Succeed)

TeejayVanSlyke

Table of Contents

You're about to embark upon your first custom software build, but you're terrified at the breadth of terminology and wary of consultants nickel-and-diming you. Or maybe you've engaged a software consultant before and are jaded from the experience, hoping that maybe your next experience won't leave you feeling burned, helpless, and penniless.

You know full well a custom software solution—a web application, a mobile application, or an internal enterprise tool—could really boost your productivity and increase revenue. You've done the numbers. You've read your customer feedback. But you're still anxious and confused by the process.

You probably know that, had your failed software project succeeded, it would have resulted in:

- More customers
- More revenue
- Less tedious work
- Less overhead

You know all of that, but still don't act. You're paralyzed by missed deadlines, overrun budgets, runaway consultants, and changing requirements.

What if there were a way—a framework—for succeeding in software projects every time? What if before you begin your next project, you knew:

- Your consultant has your best interest in mind
- You're going to walk away with software that adds value to your business
- You will be in control of your project's destiny

This book is designed to offer just that framework. I hope you'll use it as a companion to your hiring and discovery process on your next software project, helping to inform your next decisions and to empower yourself along the way. If you do, please email me. I'd love to hear how you've applied these techniques to your own project.

— Teejay

Why software projects fail

We've all heard the horror stories. Missed deadlines, overrun budgets, and buggy software. Dissatisfied clients and frustrated consultants. At the conclusion of every failed project, a retrospective analysis might cite using new, untested technology as the reason for failure. Or maybe, they'd say, the project went over budget because the designer wasn't a good fit. Or maybe ...

Technology and personnel are easy to point to as reasons for failure because they're something you can change with the flick of a wrist. Want to change technology? Pay someone to rewrite your application. Need a new designer? Fire the old one and hire another.

But typically, technology and personnel problems are mere scapegoats for a more sinister underlying problem: Inadequate communication.

When I was just 21, I was hired to lead an ambitious project for a multinational language learning company. I was brought onto the project after development had already begun. Many architectural decisions had been made that were difficult or impossible for me to question.

What's more, my point of contact at the company who was to steer development (called the *product owner*) was ill-equipped to approve or deny the decisions I was making. It's not that they were incompetent, but that they couldn't convey to me an articulated vision of what we were to build.

From a technology perspective, we were well-equipped. We had a team of top-tier engineers and were using the cutting-edge technology at the time (Ruby on Rails).

And from a personnel perspective, we were also fortunate. The company poured its resources into the project. We'd engage in daily meetings, discuss partnerships with other powerful firms, and scope out large budgets for the project.

But through it all, the common thread that finally unraveled the project was simple: There was not mutual consensus and understanding about the project's outcome and purpose. Being that I was a rookie consultant at the time, I didn't have the foresight or social capital to change course. And so, the project failed.

Software is 10% code

The most difficult part of a software project is not the programming. It's not scaling, nor is it staffing. The most difficult part of a software project is communicating why you want to build the product you're building, and in turn, understanding the technical implications of doing it.

If you tell a consultant you need to build a customer relationship management tool, they're going to have their own idea of what that means, based on their prior experience using customer relationship management tools. Their experiences will produce an impression which might be radically different from yours. And if you're not careful, you'll blindly accept their understanding as being congruent with your own.

And too, consultants ought to tread lightly when assuming you understand their techical jargon. If you tell a consultant you understand the technical implications of what you're building, it's likely there are things you don't actually understand *but also don't know you don't understand*.

It is in this flurry of mutual misunderstanding that tensions start to mount. It might start with a missed deadline. You assumed they understood you needed feature X this week, but they imagined it wasn't as mission-critical as it was. You understood that implementing feature X would be easy, but they actually assumed you knew it might be tricky.

Custom software is a service, not a product

When you buy an iPhone, your purchase comes with a list of expectations.

You expect that the iPhone live up to a basic standard of reliability: It should turn on, it should have decent battery life, it should make phone calls, it should connect to the Internet, etc.

If any of these expectations aren't met, you'll feel disappointment. Your expectations were shattered by a reality that didn't match. You might take the iPhone back to the store to get a refund or exchange, hoping to get more of your expectations met.

The same principle applies to geting your car's oil changed. When you go to get your oil changed, you expect that when you drive your car away, you'll have a new tank full of fresh oil in your car. If you drove away and a couple thousand miles later your engine started sputtering because the oil wasn't changed, you'd find a new auto shop.

This is a fair expectation because an oil change is a *commodity service*. Each instance of the service rendered is, within reason, similar enough to guarantee the same price and time to delivery every time.

But building custom software is a *creative service*. Built into the engagement is a widely-varying amount of risk on both sides of the table.

For you, the product owner, you're putting your money, time, and social capital on the line hoping to deliver a product that has a positive return on your investment.

And the consultant is risking their time, social capital, and the opportunity cost of doing business elsewhere to build something they've never built before.

For both parties, there's a high percieved risk of failure. Unlike providing an oil change, when a consultant provides software consulting services, they're wandering into the unknown. They might know the tools they'll use along the way, but the product you're asking them to build might require of them a dramatic amount of learning and discovery.

That's why it's critical to walk into the software consulting process with the mindset that you're paying for access to your consultant's brain, as opposed to paying them to build you a product.

Building software is not transactional

The difference between a great software project and a lousy one is the time and attention you're willing to devote.

I've had prospective clients in the past who demonstrated that they were unwilling or unable to provide the time and attention necessary for me to understand their business and its unique needs. And it's not their fault; they've been conditioned to believe that when you buy a service, they're freeing themselves from the burden of dealing with it.

That might be the way it works when you hire a housecleaner or a landscaper. But custom software isn't transactional. It's relational.

Unlike housecleaning, building custom software is an extension of your business. It's the codification of your years of hard work and understanding. In a way, your custom software build is actually a digital representation of your brain.

In order to succeed building custom software, you'll need to be available to answer questions. You'll need to devote time and attention to your project every single day. You'll be asked questions that, to you in your line of work, might seem like common knowledge. And it might exhaust you. Take your exhaustion as a sign your consultant is doing great work.

How to find and hire a trustworthy consultant

I was at a local tech meetup. A man approached the table. "Is this Tech Tuesday?"

He was seeking a consultant who could help him finish development on his mobile app. He'd been through the ringer, having contracted with multiple shops and each time having a terrible experience.

Feeling down and out, he just wanted to find development talent he could trust to deliver on their promises.

I spent the better part of an hour listening to his needs and providing insight into how he might go about finding the right person. And I wanted to ensure he didn't waste any of his precious time and money on the *wrong* person.

How do you find a consultant who has your best interest in mind? Here are some of the things to look out for:

Do they have the heart of a teacher?

Do you believe with your heart and soul that your candidate has your best interest in mind? Is their demeanor more like that of your kindergarten teacher, or a used car salesman?

Software development is a two-way engagement. It's tempting to assume you can hand a consultant a pile of requirements, wait six months, and get what you paid for. But in reality, your relationship with them is collaborative. Your understanding of how they build your application will prove valuable to you long after their engagement ends.

Ask them to explain how they would solve one of your core problems. And continue to ask questions until you understand, to the point where you could explain their explanation to someone else. If they're reluctant to engage with you in this way, run. Fast.

Are they willing to challenge your assumptions?

Often, the software you think you need to build isn't the most expedient way to your business goals when technical realities smack you in the face.

This doesn't mean you ought to find someone who is lazy or unmotivated. But you want them to push back when you explain your needs, and to justify their concerns.

Numerous times I've told my clients that the feature they thought they needed next probably didn't need to be built yet. Or that there was a way to do it that would better serve their needs for less cost.

Your consultant is a running expense to your business. Finding one who treats themselves that way is critical.

Are they an effective communicator?

The end result of poor communication in a software product is a poor product, no matter how awesome your consultant's technical skills might be.

If you hire a remote consultant, it's critical they have excellent written communication skills. Their work schedule might not overlap with yours to engage in phone calls and chats. And you don't want them to rely on phone calls because then they're burning all your money squawking on the phone instead of building software.

What you're really looking for is someone who can effectively engage you through your project management software. Someone who can answer your questions thoroughly, follow up regularly, and ask questions consistently.

When you're getting to know your candidate, email them with a question about a specific aspect of your project. Get a feel for how they engage in an asynchronous (i.e., not realtime, like a phone call or chat) environment.

On one extreme, you might find their responses thoughtless and overly brief. They might not satisfy your need for clear answers.

On the other hand, they might be too verbose, explaining things that needn't be explained and ultimately confusing you.

Give them the benefit of the doubt, but ask yourself whether their manner of written communication empowers you to make effective decisions about your product. Did your test exchange leave you feeling more confident, or more confused? How would you feel if you were relying on them to put out a fire in your business? Would you have faith in their execution?

Ask them to explain a time they failed. Then ask how they resolved it.

Every consultant has failed at some point in their career. They've botched a client deadline. They've built the wrong feature. They've deployed buggy code to production. And they've probably indirectly or directly lost their clients customers because of those mistakes.

If anything is certain in software development, it's that you'll fail. Because of this, the critical piece when vetting a consultant is finding out how they respond to failure. Looking for a consultant who has never failed will turn up two kinds of consultants: Those who have failed, and liars.

Instead of facing the lost cause of finding a consultant who always succeeds, ask them how they've responded to failure. Do they step up to address problems irrespective of who was to blame, or do they focus on placing blame, whether on their managers, their clients, or the technology?

Failure isn't always their fault, but the attitude with which they approach failure is an indicator of their character. Use this as a metric for how they might react to an issue on your project.

Do they have professional references? Call them.

Always ask for the phone number of at least one professional reference. It might be a former manager, another client, or a colleague.

When you call, explain you're thinking of hiring your candidate, but that you have a few questions and are hoping they might be able to help. If they had a good working relationship, you'll hopefully find them eager and willing to help.

Don't know what to ask? Try these questions:

1. What obstacle would have prevented you from hiring/working with them?
2. What value did they provide to your business?
3. Would you recommend them to others?

These specific questions will help you identify:

- If there was something that gave them pause before hiring the candidate
- What specific value the candidate provided to a real-life business
- Whether someone would go out of their way to recommend the candidate's work

Now you have a first-hand testimonial with real talking points upon which you can base your decision!

Get a second opinion from someone you trust.

If you know someone else in the industry whose opinion you trust, why not hire them to help you with the process? A seasoned consultant or technical manager will have years of experience with other consultants and will know what to look for.

They'll be able to review the candidate's sample code and portfolio to tell you whether they'd feel comfortable working with them.

And most importantly, they'll give you an impartial opinion because you'll pay them a fee for helping with the hiring process—not for being a prospective candidate.

Why you should ask for—and pay for—a technical discovery

If you've paid for a custom software product that went over budget, you know the pain that comes when you start writing checks beyond your budgeted amount.

It's easy, after the fact, to name reasons why your project went over budget and was delivered late. But often, these problems can be mitigated upfront by taking the time to get an impartial opinion before you start paying a consultant for actual development. This process is called *technical discovery*.

What is a technical discovery?

A technical discovery is the activity of identifying the business requirements, technical landscape, and other factors that contribute to the overall cost of your project.

Typically, a consultant will engage you for one or more intake sessions, wherein they will ask you a series of questions pertaining to the nature of your business, the product you're hoping to build, and the outcomes you hope the product brings. Then they'll analyze what you've told them through their years of technical experience to deliver to you following in a document:

- Technology and architecture recommendations
- User stories
- Cost estimates
- Caveats and concerns

Technology and architecture recommendations

Your project, like every one before it, was similar to previous projects your consultant worked on and different from other ones. A good consultant will be able to make, based on their experiences, recommendations of what technologies fit your project.

Typically, they'll provide you with a brief summary of their recommendations, as well as the analysis that led them to their conclusions.

User stories

A *user story* is a way to describe a software feature from the end-user's perspective.

What this means is that, rather than explaining the feature in terms of the technologies employed and the systems utilized, a user story puts the user at the focal point.

A well-written user story has 3 parts:

- Actor ("As a...")
- Feature ("I want to...")
- Purpose ("so that...")

For instance, if you had a feature that sent your user an email with a report each time you pressed the "Send Me A Report" button, you might write the user story thusly:

> As a user, I want to receive a report via email when pressing the "Send Me A Report" button so that I can read the valuable report later.

The reason we divorce user stories from their technology implications is to reinforce the fact that we cannot make low-level ("in the weeds") assumptions about the technical aspects of your application before we've built it.

This might sound alarming. "But I'm hiring expensive experts to build me a product. They should know how they're going to build it from the beginning!"

While that assumption is valid for a *commodity service* like our oil change example, it actually works against your favor to make assumptions about low-level technical details early on in your project.

You see, your business, and every business, is in a constant state of flux. The solution you think will work today could be rendered obsolete or irrelevant tomorrow. And when it does, the technical analysis around that solution will be rendered obsolete right with it.

Instead of seeking to understand and evaluate every nuance of your product from the start, a good product discovery takes a bird's-eye approach, being sure to consider each feature carefully, but only insomuch as to produce mutual understanding between you and your consultant.

The beauty of user stories is that, even if you take your discovery document to another consultant for implementation, their analysis will be relevant no matter your new consultant's opinions about technology.

Caveats and concerns

This is where a good consultant will voice all of the reasons you might consider reducing your scope, or walking away from the project altogether. The technical implications of the product you want to build might, in the eyes of your consultant, involve a high amount of risk to you or your business. Maybe an integration you want to add isn't well-documented. Or maybe there's a feature that has a high chance of failing or being prohibitively expensive.

Cost estimates

And of course, the moment you've been waiting for: *How much is this thing going to cost me?!*

This is where the emotions start to run wild and questions start piling up. After all, estimating and bidding on a custom software project is difficult, contentious, and confusing.

Fortunately, it doesn't have to be that way. As we'll see in the next chapter, understanding your consultant's frame of mind when they estimate your project will help you take control of your project and build the best product for your money every time.

Don't go chasing waterfalls

There are two main opposing schools of thought in software development, and you might want to ask your consultant how they conduct discoveries before you begin so you understand what you're in for:

Waterfall development is where a full specification is meticulously outlined upfront, every corner of the application build is thoroughly examined and blueprinted, and then that document can (theoretically) be handed to any developer and they'll be able to build the thing from scratch without communication from the stakeholders.

Agile development rejects the chief claim the waterfall school makes: that it is possible to know everything about a software project upfront. Agile recognizes that requirements change based on new conditions, and that speculation about an application build upfront are subject to change as a result of new technical limitations, business condition changes, and cost reappraisals. Because of this, an agile discovery offers the highest value at the lowest cost.

A waterfall-style discovery does a disservice to you and your business. It makes claims about timelines, technology, and user interface which are grossly inaccurate, because all of those things will change throughout the course of development as a result of new

findings through prototyping and experimentation.

Why pay for discovery?

If you asked for a price quote from the moving company and they told you you'd have to pay for it, you'd probably hang up the phone.

But when you're building a complicated custom software product, paying your consultant to estimate cost is in everyone's best interest. Let me explain.

As we discovered earlier, building custom software is a *creative service*. From the consultant's perspective, unbilled time spent scoping and analyzing your project is time they could have spent making money. This means that, in spite of their best efforts to the contrary, it's likely they'll spend as little time as possible on a free analysis. It's also likely they'll want the outcome to be that you hire them, even if it's not in your best interest.

People respond to incentives, and your consultant is no different. Finding a consultant who offers a paid technical discovery means that rather than treating you like a sales lead, they'll treat you like a customer. You become a priority in their day's work, instead of an expense.

And on top of that, a consultant who knows they're being paid for discovery will be brutally honest with you when it matters most.

Imagine engaging with a consultant for an unpaid discovery session when they're desparate for work. They might tell you the project will take 10 weeks at their weekly rate of $1,000. To you, that means your project will cost $10,000. You sign a contract with them to engage them at that rate.

They told you what you wanted to hear because they needed another client, and fast. And get this: Because they weren't billing for the time spent on your discovery, they didn't do enough analysis to properly scope your project. The project overruns, and takes 20 weeks.

Paying for technical discovery is a hedge against the possibility that your idea is more costly than the value it produces. It protects you, the busy business owner, from the hardship of being sold a project that was never going to be completed on time in the first place.

It costs what?! How to understand your consultant's estimate

At the end of your paid discovery, your consultant will deliver to you some sort of price estimate or bid.

Some consultants might provide a flat fee for the entire project. Others will provide an hourly or weekly rate with an estimated number of total hours.

How can you dissect these numbers to understand their quote and make it work in your favor?

Fixed prices versus estimates

If your consultant gives you a fixed fee, you'll be tempted to ask them for their hourly rate, divide the fee by that much, and see how many hours they think it'll take them.

For instance, if their final bid on your project is $10,000 and you find out their normal hourly rate is $100, you might assume your project will take them 100 hours.

Don't do this. When a consultant gives you a fixed price quote, they're insuring themselves against the chance that they've made a grave error in their estimation. A good consultant will pad a fixed price by 15%, 25%, 50%, or sometimes even 100% of their initial estimate analysis. In our example, there's a good chance your project might only take 50 hours.

In providing you with a fixed quote, they're opting to absorb your risk. They're saying they'll deliver to you fully working software for a fixed price, so you don't have to worry about the possibility of cost overruns. In order for the consultant to protect themselves, they need to assume the worst.

Depending on the size, scope, and variables in your project, a reliable consultant will tell you they cannot offer a fixed price quote because the complexity of your project is too great to be able to accurately assess a price they're comfortable offering.

Instead, they'll offer you an estimate in terms of time to delivery and cost per unit of time. Your consultant will bill on an hourly, daily, weekly, or monthly basis and provide you cost estimates based on that interval.

Estimates are just that: estimates. Your consultant, based on their careful technical analysis of your business requirements and their years of experience building projects similar to yours, will deliver to you a feature-by-feature breakdown of cost:

Feature	Estimated # Weeks	Estimated Cost
User Signup	1	$1,995
Setup Wizard	3	$5,985
Dashboard	2	$3,990
Total	6	$11,970

In the example above, the consultant has given us an estimated number of weeks to deliver each of our features, at a price of $1,995 per week.

Why you should take the estimate and not the fixed price

The idea of embarking on a project whose cost is unclear and subject to change can be daunting. As a business owner, how can you plan your expenses and project whether your investment will yield a return?

Because of this, you'll probably be tempted to go with the fixed price quote if one is available. While you effectively pay a fee for price stability, you're more able to plan for tomorrow when you know the most you're going to pay for your software product.

But it doesn't usually work out that way. And what I'm about to tell you might change your mind the next time around.

Fixed price means a fixed scope

If you've been a part of a custom software project before, you know requirements are subject to change during the course of development.

When you accept a fixed price bid for a fixed set of features, you've jeopardized your ability to respond to these changes.

Once I completed the functional requirements for a fixed price project for an agency, and sent the prototype to them for review.

They agreed that what I built matched their original expectations perfectly. But their client's wishes changed, and they wanted some additional enhancements to the original scope. We had to draft another contract with the enhancements, costing us valuable

development time spent managing paperwork.

With a fixed price structure, your project is more or less set in stone. You're unable to respond to your business's changing conditions in the middle of development because you already agreed to a fixed feature set for a fixed price.

Because of this, there's a chance the product your consultant builds you won't be the ideal product for your business by the time they're done.

Who decides when the project is done?

You've completed a technical discovery. You have user stories in hand. They're stated in the contract. But what happens when your impression of a feature and their impression of a feature are slightly different? What happens when, during the course of your discovery process, you and your consultant neglected to discuss one key factor in one of your features that could make or break your business?

And what if you thought they understood that, but they didn't, and it's nowhere to be found in your contract?

The truth is, the only analysis that captures all of the requirements, risks, and technical architecture knowledge required for building a software product is *actually building the software product*. So, short of asking your consultant to do that in lieu of your technical discovery, your best case scenario is to take your consultant's hourly or weekly rate and guide the project to completion yourself.

And as we'll see in the next chapter, it's easier than you'd think to take control of your project's destiny each week. You just have to...

Own your project like a boss

The contract is signed. You and your consultant are amped and ready to start in on your new application. Now you wait. The project's out of your hands. You have to let chance decide your fate.

Or you can own your project like a boss.

No, I don't mean like a pointy-haired boss. More Like A Boss.

You see, as the product owner, you have an opportunity to guide development you might not realize.

Before you sign your next contract, you're going to make sure of two things:

1. You both agree, in writing, to engage in an *iteration planning meeting* once every week or two weeks, where you'll decide the work you want to attempt to complete for that *iteration* and generate *points-based estimations* for each.
2. The consultant will deliver you, at the end of each iteration, their *velocity*.
3. Your contract stipulates either party is free to walk away from the engagement with no more than 14 days' notice.

Iteration planning: Your command center

Signing a check for tens (or hundreds) of thousands of dollars is daunting. More daunting is waiting around for your consultant to build your product, not knowing its status and fearing that they're making assumptions that don't match your business's needs.

What if there were a way you could take back that control? What if every week or two, your consultant showed you real progress on your product, and you were able to dicate to them what should be built next?

That's the beauty of *iterative development*. Instead of spending the equivalent worth of a nice car and handing your consultant the keys, you get to sit in the driver's seat.

Each week or two, your consultant will do the following:

- Review progress on your project in the past iteration.
- Calculate the *points* completed last iteration, along with your *velocity*
- Pick *user stories* from your *backlog* to complete in the next iteration.

- Estimate how many points each user story will cost

Points: They're not hours and they're not dollars

Estimating software in terms of dollars or hours is woefully inaccurate and misleading.

That's because we humans are absymal at estimating how long things take. Do you know how long you spent, to the hour, on the last medium-sized task you completed?

If we have such a hard time conceptualizing time spent on activities we've already completed, it's no wonder we're so terrible at estimating the time it'll take to complete tasks yet to be completed—especially when those tasks carry a high amount of uncertainty.

Fortunately, there's a better way.

Instead of estimating how long or how much, they'll estimate *how difficult*.

Imagine a task you know you can complete in hardly any time at all. Let's say ... deleting an email. We'll assign this task 0 points.

Okay, now imagine a task you know you can complete in less than an hour. Or maybe, in a bit more than an hour. It doesn't matter. What matters is that you know you can complete it without trouble, and you have little to no uncertainty in the matter. We'll assign that task 1 point.

Now imagine a task twice as difficult as that. That task gets 2 points.

And one twice as difficult as that? 4 points. See a pattern?

Okay, now imagine a task that's really complicated. It has a lot of moving parts. You might even be able to dissect it into multiple tasks. That task is probably somewhere around twice as difficult as the 4-point task, right? We'll give that 8 points.

When your consultant provided you their initial estimate, they likely evaluated each of your user stories by assigning them a point value. The cost associated with point values is arbitrary; what's important is the relationship of one estimate to others.

Each iteration, you and your consultant will look at the work completed that iteration. Because your consultant assigned each user story a point value based on their assessment of its difficulty, you can sum the point values for all your completed user stories to determine your *velocity*.

Velocity: Your project's speedometer

If points communicate the difficulty of each user story, then we can determine how much work was completed by summing all of the points for all of the completed stories for the iteration.

Imagine that, in the past week of work, your consultant has completed the following stories:

User Story	Points
As a user, I want to be able to add another user as my friend so I can communicate with them	2
As a user, I want to be able to send a message to my friend so I can communicate with other users	4

Your consultant completed 6 points last iteration, so your current velocity is 6. This means:

- You can plan approximately 6 points worth of work to complete next iteration.
- You can use this number, along with estimates for future work, to approximate cost and time to delivery of the remainder of the project.

If your consultant completed 6 points worth of work last iteration, and you have 48 points worth of stories remaining in your backlog, you can estimate that 8 weeks of development work remain at the current velocity (48 / 6 = 8).

Multiply that figure by your consultant's weekly rate, and you'll have an estimate of remaining cost.

The beauty in this approach is that each week, you're able to analyze cost based on real-world data. You're not making assumptions based upon how much your consultant blindly thinks they can complete, but are making projections based on what they've already finished.

Freedom to walk away

But what if you don't feel like you're getting adequate value for your money? Had you accepted your consultant's fixed-price contract, you'd be stuck in the contract until its successful completion. They might take their time to complete your project, or worse, go dark on you entirely.

When you engage your consultant on a time basis with loose termination terms, you're able to walk away from a toxic relationship before it bites you and your business. Because you're paying your consultant on a short interval, you're free to exit the relationship and still retain the value they've already delivered you.

By taking the reins of your project and owning its execution, you'll be less prone to being taken advantage of and more equipped to respond to changes. And a consultant who has your best interest in mind wants this. They want to know that you're getting tremendous value for your money, and want you to be able to find that value elsewhere if necessary.

A special offer to you

So now you understand why software projects fail, and how to make yours succeed. You know how to find and hire a qualified consultant, why you should pay for a technical discovery, how to interpret your consultant's quote, and how to take control of your project week-by-week.

You're about to go out into the wild and apply what you've learned. And I want to offer my help on your journey.

Have a question about terminology or want a second opinion? Email me and I'll help however I can

Have a success story to share? Email me and I'll add it to the book

Just want to say hello? Email me (Notice a trend?)

Glossary

Agile development

A methodology of software development which recognizes that requirements change based on new conditions, and that speculation about an application build upfront are subject to change as a result of new technical limitations, business condition changes, and cost reappraisals.

Backlog

A list of user stories remaining to be implemented that are not part of the current iteration of work.

Commodity service

A service that is easily replicable, doesn't require high-touch creative processes, and carries a low amount of risk to deliver. Examples: housecleaning, oil changes, pizza delivery.

Creative service

A service that is difficult to replicate and carries a high amount of risk due to the uncertainty involved. Examples: programming, design, engineering.

Fixed price / fixed bid

When a contract stipulates that its deliverables will be delivered in full upon receipt of a fixed amount of money. Contrast to time-based billing, wherein the product owner pays for the consultant's services for a set period of time, regardless of outcome.

Iteration

A set period of development time, after which the work completed will be reviewed and priorities realigned.

Iteration planning meeting

A meeting, usually weekly or bi-weekly, between the product owner and software consultant(s), to review progress made for the iteration and scope work for the next iteration based on the current *velocity*.

Points-based estimation

A method of estimating task difficulty relative to the perceived difficulty of other tasks.

Technical discovery

The activity of identifying the business requirements, technical landscape, and other factors that contribute to the overall cost of your project.

User story

A way to describe a software feature from the end-user's perspective.

Velocity

The average number of points completed per iteration, for all past iterations.

Waterfall development

Development methodology where a full specification is meticulously outlined upfront, every corner of the application build is thoroughly examined and blueprinted, and then that document can (theoretically) be handed to any developer and they'll be able to build the thing from scratch without communication from the stakeholders.