Why Software Projects Fail

and 6 secrets to help yours succeed
WHY SOFTWARE PROJECTS FAIL

6 Secrets

SECRET 01
FORGET TECHNOLOGY
Your project should start with questions that have nothing to do with technology and everything to do with your business.

SECRET 02
REMEMBER THE HUMANS
The most innovative software solution will fail if no one uses it. Your developer should gather user requirements through detailed questions and scenarios.

SECRET 03
KEEP THE CONVERSATION GOING
Change is inevitable as software develops. Pick a developer who manages change through structured, ritualized communication.

SECRET 04
THINK SMALL
Large projects are virtually guaranteed to fail. Work with your developer to keep a lean team, limit scope and avoid Feature Creep.

SECRET 05
TRY IT BEFORE YOU BUY IT
You’d never buy a car without a road test. Choose a developer who supplies a working prototype you can try out before investing in the final build.

SECRET 06
YOU CAN’T AFFORD NOT TO TEST
Think testing is expensive? You haven’t heard of Healthcare.gov.
Failure is an option. (SORRY, HOUSTON.)

Failure is actually more than an option for some software projects. It’s a given.

Why not just open a window and toss out bags of cash? It would be a quicker, less painful way to lose insane amounts of money. With such daunting odds, why would anyone risk developing custom software?

Simple: If you want your business to grow, at some point off-the-shelf, one-size-fits-all software won’t do the trick. You’ll need software tailored for your specific business needs, goals and, most important, your end users. These may be your employees, who will expect it to make their jobs easier. They may be your customers, who will expect an enhanced customer experience. If your end product disappoints the humans who use it, it will be a failure, even if every function and feature works perfectly.

Besides, you can beat those odds. We help companies do it every day. This guide will give you six secrets to keeping your project on time, on budget, focused on users and headed for success.
One of the biggest failure factors in software development has nothing to do with technology. It’s lack of business analysis before the development begins.

Analysis means asking questions, and over my company’s 15 years I’ve discovered that the best questions are nontechnical.

Say the client is a widget maker who comes to us asking for a product to help with inventory management. I’ll ask why the company makes widgets. I’ll ask about the challenges of the widget market. I’ll ask what the client hopes the business will look like in three to five years. Does he or she just want to sell more widgets? To expand into other product lines? To disrupt the widget industry with a new uber-widget?

The answers may have nothing to do with inventory management, or they may have everything to do with it. Maybe a lack of effective inventory management isn’t the problem at all. Maybe it’s just a symptom of a larger, unaddressed challenge that is keeping the business from meeting a greater unrecognized potential.

Often we create the product the client described initially, but sometimes the answers lead us in a different direction altogether. Read about our client, First Crush Tastings, in the sidebar.

**THE TAKEAWAY**

Be patient if your developer asks probing questions, even though they may seem irrelevant to the project’s goals. In fact, be pleased! Thorough business analysis at the beginning of a project is the best way to assure success at the end.
The first questions should be wide-ranging and probing. The next questions should be narrowly focused, but equally probing.

Your developer will use them to create the dozens or even hundreds of specific software requirements included in the project. You’d be surprised how much complexity can hide inside the simplest-sounding request.

Say a client wants a login page. With no further questions, the developer builds and launches a basic login page.

**THEN THE PROBLEMS START POURING IN:**

⚠️ Users are sharing credentials, so unauthorized staff is accessing restricted areas.
⚠️ Users who forget their login credentials can’t figure out how to retrieve them.
⚠️ Users don’t have to agree to terms of service before accessing the site, and corporate lawyers are tearing their hair out.

Can these problems be fixed after launch? Sure. Is it more expensive? You bet. It’s way easier and cheaper to zero in on the right requirements first than retrofit later. Your answers to requirement-gathering questions should help your developer:

**1. UNDERSTAND THE PAIN BEHIND THE REQUIREMENT**

Why is this feature desirable? How is the business coping without the feature today, and how will the new software improve things? Before building a digital solution, developers need to understand the analog, aka human, process.
2. ELIMINATE LANGUAGE AMBIGUITY

Human language is ambiguous; software code is not. For instance, by asking for a “login form,” clients sometimes mean they want to charge a subscription to access premium content. Exact meanings must be pinned down.

3. IDENTIFY COMER CASES

Human behavior is often as unpredictable as human language is ambiguous. In software terms, an action taken outside of typical, predictable operating patterns is called a corner case—for instance, sharing login credentials for concurrent access. These will require additional features.

4. WRITE USER STORIES

To collect and organize the relevant information from the Q & A sessions, we use a Mad Libs™-style process called the User story. The basic blank User story looks something like:

“As a (user type), I want this feature to enable me to be able to (task to be performed) so that I can get (resulting business value).”

Once the phrases are filled out, save the stories to your project management system.

5. CREATE A DEFINITION OF “DONE.”

When the desired features are identified, we “translate” every User story into a How-to Demo, or HTD. This show-don’t-tell methodology lists the steps we’ll take to fulfill the requirement and removes any remaining ambiguity. It also creates a definition of “done”: When these features are enacted, following these steps, the agreed-upon requirements will have been met.

TAKEAWAY

Get the requirements right through probing questions, precise answers and walking step-by-step in a user’s shoes.
The goals are clear, the requirements are nailed down and the project is well under way. Then suddenly there’s a curve ball. A new government or industry regulation takes effect, one your software needs to address. Your company brings a new CEO onboard, and he or she has a different vision for the project. In software development, change is the only constant. The last thing you want is a developer who fights that truth. You want one who embraces change but knows how to manage it.

Like other Agile practitioners, we manage change through communication. I don’t mean random conversations, but face-to-face, regularly scheduled interactions. I know that meetings have gotten a bad rap as time-wasters, but we find just the opposite is true. Talking with each other during frequent, recurring interactions empowers clients and developers to respond to change quickly and efficiently.

We meet with every client weekly or bi-weekly to discuss status and consider any change factors that may have surfaced. Our teams have daily stand-up meetings, where members share their progress and coordinate efforts. Team members also review one another’s work as a matter of course, reducing opportunities for error. We do this because it works. We also do it because the cost of not communicating is staggering.

The cost of not communicating is staggering.

THE AGILE MANIFESTO
In 2001, a group of software pioneers introduced a responsive, people-centric new approach to product development. They called their doctrine the Agile Manifesto, and it has become a mainstay in public and private sector software teams around the world. According to The Standish Group, “software applications developed through the agile process have three times the success rate of the traditional waterfall method and a much lower percentage of time and cost overruns.”

THE FOUR MAIN AGILE TENETS WE VALUE ARE:

1. Individuals & interactions over processes & tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan
ACCORDING TO A RECENT STUDY BY THE PROJECT MANAGEMENT INSTITUTE (PMI):

- “Companies risk. $135 million for every $1 billion spent on a project and new research indicates that $7.5 million of that $135 million (56 percent) is put at risk by ineffective communications…”

- “Ineffective communications is the primary contributor to project failure one-third of the time, and had a negative impact on project success more than half the time.”

- “High-performing organizations (those completing an average of 80 percent or more of projects on time, on budget and within goals) create formal communications plans for nearly twice as many projects as their lower performing counterparts (which complete fewer than 60 percent of projects on time, on budget, and within goals).”

TAKEAWAY

Ritualized communication is the most effective way to manage change and move the project forward. Without it, you’re talking failure.

NASA’s $193,000,000 Failure to Communicate

What can happen when development teams don’t talk to each other? A cosmic flameout. In 1998, NASA’s $193-million Mars Climate Orbiter approached its target planet at the wrong angle and burned up in space. Among the engineering problems causing the calamity: The thrusters were more than four times too powerful.

The mistake happened because different members of the engineering team had used different units of measurement. One group measured in English units of pounds-force seconds. The others used metric Newton-seconds.
IN 1990, THE STATE OF FLORIDA STARTED WORK ON A FEDERALLY FUNDED INITIATIVE to automate its child welfare information system. The cost estimate was $32 million; the target completion date was 1998. With a team of 109, plus 3 IBM consultants, the project finally finished in 2005 at a cost of $230 million. The consultants’ fees alone totaled $1.8 million a year.

IN 1999, UNDER THE SAME FEDERAL INITIATIVE, Minnesota started its own state child welfare information project with a lean 8-person team. By limiting the scope of the project, the team completed it in 2000 at a cost of $1.1 million. The software performed essentially the same tasks as the one in Florida.

THESE ARE TWO OF COUNTLESS EXAMPLES PROVING THAT IN SOFTWARE, BIGGER IS NOT BETTER.

The 2013 Chaos Manifesto, published by the Standish Group, states that large projects-defined as $10 million or more-have “virtually no chance of success: ... more than twice the chance of being late, over budget, and missing critical features.”

On the other hand, small projects-defined as $1 million or less-have a 7 in 10 chance of successful delivery.

Large projects have virtually no chance of success.

It makes sense. As the Chaos researchers point out, fewer people on a team make communication and rapport-building easier. A limited project scope increases the likelihood of accurate time and cost estimates because there are fewer variables. You can usually break down large projects into smaller, more manageable iterations, but watch out: Even if your project starts out small, there is a big, bad wolf lurking in every software forest. It’s called feature creep, and it will try to fatten up your project with expensive features nobody will use. Then it will swallow your budget whole.

SECRET 04:

Think Small
Keep the wolf away from your project’s door by applying the 80/20 rule (also known as the Pareto Principle), which maintains that 80% of an event’s results come from 20% of the causes. In software terms, this has been interpreted to mean that 80% of a product’s value comes from 20% of its features.

To get the biggest bang for your buck, focus on the 20% users actually want and will use. Which features are those? Review user analytics you can find, or bring in groups of end users and ask them.

**TAKEAWAY**

Software projects with limited scope and only the most essential, user-focused features have the greatest chance for success.

**WHAT FEATURE CREEP LOOKS LIKE**
WHY SOFTWARE PROJECTS FAIL

BROOKS’ LAW: THE MORE, THE NOT MERRIER

When a project is running late, it’s only natural to think adding more people will hurry things up. But according to Brooks’ Law, introduced by software engineer Fred Brooks in The Mythical Man-Month, as you add developers to the project, the “conceptual integrity” of the whole system is violated.

Adding manpower to a project in fact increases time and cost

Due to the interconnectedness of a software system, this results in technical debt and a higher incidence of bugs. More than anything though, adding more devs to the effort compounds the amount of communicating the team must do to make their sub-tasks handshake. More communicating means less coding and increased risk of miscommunication.

The counter-intuitive result is that by adding manpower to a project, you in fact increase the amount of calendar time (and cost) that project will require.
You wouldn’t consider buying a car without going for a test drive. It’s the only way to get a feel for the vehicle. How smooth is the ride? How quick is the pickup? How’s the leg room, elbow room, dashboard interface? Are there are enough cup holders?

Yet many software developers do not routinely provide a working prototype for clients to try before they invest in the system build. And that complex engineering carries a much heftier price tag than the most luxe set of wheels.

ADVANTAGES TO PROTOTYPING:

1. More stakeholder involvement in development
2. A better user understanding of the system
3. Earlier detection of program errors
4. Identifying confusing or missing functionality

Our fully functional, interactive prototype, ClickModel, validates stakeholder requirements (See #2) and uncovers some that might have been left out or incompletely addressed. It reduces risk and enables us to deliver a reliable estimate of production cost and duration. It’s a proving ground for ideas and the least expensive time for experimentation, sometimes surfacing hidden opportunities to boost ROI.

A HOLISTIC APPROACH TO PROTOTYPING

Successful prototyping encompasses these five key, interdependent, areas of focus:

- **Strategy**: Business objectives, user needs
• **Scope:** Functional specifications, content requirements
• **Structure:** Interaction workflows, information architecture
• **Skeleton:** Interface behavior and layout, navigation behavior and layout
• **Surface:** Visual design of interface elements; visual design of text, graphics and navigational components

Best of all, ClickModel provides an artifact that stakeholders and developers can gather around to communicate about the user experience and reach agreement about features and scope. Suddenly a process that started in the abstract with an idea becomes concrete. And when it’s your idea that at last comes alive, we challenge you not to get excited.

**TAKEAWAY**

To ensure the most successful user experience and validate requirements, there is no substitute for working prototype.
When budgets (and tempers) are frayed and time is running out, testing can be an easy target for the budgetary axe. However, as Delta Airlines knows too well, the cost isn’t in testing. It’s in not testing.

In December 2013, an undetected software bug created an incorrect display of rock-bottom Delta fares. A round-trip ticket from North Carolina to Hawaii was listed as $180 instead of $1800. For a Chicago family of five, tickets to visit relatives in Idaho cost $240 total, or less than $50 apiece. United Airlines felt Delta’s pain. That September, some of their passengers had flown for free, after zero-dollar fares found their way into the system.

In today’s complex systems, a single change in code can break the system elsewhere. Don’t risk it. Choose a developer you can depend on to release quality, bug-free code when your software launches and has a rigorous automated (computer-run) testing program in place for continued monitoring. Automation allows easier upgrades to underlying frameworks, safer code refactoring and acts as an early detection and warning system of potential problems.

SECRET 06: You Can’t Afford Not To Test

Use the checklist below when interviewing your custom software development firm to find out if their level of automated testing & deployment meets the standards that your project will need:

- Does your partner use automation?
- Do they do both behavioral testing and unit testing?
- What percentage of the codebase do your partner’s tests cover?
- Do they have development standards such as “one test per user story”?
- Does your partner release code to the production server?
- Can your partner do “backwards deployment” (moving data from production back to staging and development environments)?
- Is “passed tests” a requirement to deploy?
- Is the process automated?
Although building and maintaining the automated test suite does take time, it saves time later on. Research suggests that if a bug is found and not fixed within the first 24 hours of being identified, it can take multiples longer to resolve. You can’t have good quality without investing in good quality, and automated testing is a critical form of prevention when it comes to managing product quality.

Automated testing. Ask for it by name. And by the way: The Delta glitch was eventually fixed in 15 minutes.

TAKEAWAY

A suite of computer-automated tests before launch and later is essential to detect bugs and ensure product quality.

Not Ready for Prime Time

Industry experts point to the launch of Healthcare.gov on October 1, 2013, as a glaring example of what can go wrong in a so-called Big Bang Rollout—where a complicated project goes live in one massive single shot, instead of using a saner, safer incremental approach that includes adequate testing and limiting scope.

One dead giveaway that healthcare.gov was not ready for launch was that at go-live time the code still contained placeholder text, put there by engineers pending final language approval.

- “TODO: add functionality to show alert text after too many tries at log in”
- “Fill In this with actual content.
- “TODO: maybe modify the below to use a similar method instead”
Founded in 2000, we’ve helped hundreds of businesses scale operations and reach more people through the power of custom software built on open source.

The solutions we create often take the form of websites, mobile apps and integrated systems but we are much more than your run of the mill web development shop.

We help our clients harness the unrealized potential within their business. We don’t believe in one size fits all solutions. We do believe in listening and thinking creatively. We are a group of designers, developers and strategists who are united in our passion for helping our clients take their business to the next level.

Talk to one of our experts to learn more about what we can do for your project:

512-553-6830
info@praxent.com