

**13 Lucky Tips
to Reduce Risk
in Your
Custom Web
Project**

A mini-report
by J. Timothy King

This article is about managing a software project. More and more, average people and businesses are managing software projects in the form of customized blogs, websites, and e-commerce shopping carts. Even more so, if they are trying to deploy a custom intranet application, for internal use by their company's employees, or an extranet application, for use by their customers and corporate partners, they may find themselves faced with developing a fully custom software system.

For example, one project I recently worked on is a custom extranet application that a certain company's clients use to submit jobs to the company. As each job request goes through the system, a number of people in the company need to add information and sign off on it. This automated system was thick with business rules for this particular company and as a result was almost completely custom, because no off-the-shelf software got even close to doing the job the way they needed it done. It was expensive for them to develop, but it allowed them to automate a process that previously had been manually intensive. (Before this, they would email Excel spreadsheets around. Now, the computer handles all the grunt work, and frequently, all a human has to do is click "Approve.")

This is an extreme example, but more moderate examples also exist. For example, a [*Gilmore Girls* fansite](#) I put together years ago had custom database features in order to store memorable quotations from the show with fan commentary, linked to episode and topical guides. That's something that has not to my knowledge been done before or since. It required custom programming and configuration to process user submissions, store the data, and display the data in the right format.

Unfortunately, if you're facing this type of project, many of the consultants and service providers you talk to will not be aware of the software development issues involved and how to manage the software-development risk. They may try to shoehorn your problem into something off-the-shelf software can accomplish. Or they may just "hack together" custom programming, without managing the complexity of the underlying software design. Or they may want an up-front specification of everything your project needs to do and may want to spend months working out these details, under the guise of good planning. But how can you plan a project you have zero experience with? You can't. And neither can they.

Now, this is not a problem if all you need is a stock installation of off-the-shelf software. For example, if all you need is a WordPress blog with off-the-shelf plugins and an off-

the-shelf theme, the traditional approach is the right one. Figure out what you need, install it, make sure it works, and release it to the world. Even if all you need is a custom theme for a stock website, the traditional graphic design approach—3 mock-ups, choose one, then implement it in code—may be the best approach. Even if you have a semi-custom installation of a more complex web application, like a content management system or an e-commerce shopping cart, the traditional approach may be acceptable.

But the more customization your project requires, the more software engineering concerns will impact it, and the more ignoring these concerns will put your project at risk.

This is especially important in a financial squeeze, where you need to maximize what you get for your software-development dollar and minimize the risk that the project will go off track or bust.

Therefore, take heed of these lucky-13 software development tips in planning and executing your next web development project:

1. **Resolve to take tiny steps.** There's no reason you have to develop and deploy a large website all in one huge lump. Building software is not like building a bridge. When you build a large bridge, you can't start with a small bridge, send a few people across it to see how well it holds up, and then build it bigger to let more people cross. But you **can** build software that way. Start with top-priority features, one by one, get feedback from actual users, and use it direct your next efforts. This assures that you actually do make progress and helps guard against wasting resources going down the wrong path.
2. **Find an expert you can trust, then trust him.** Don't just look for someone to bang out code for you, but for someone to advise you on technology and on the software development process. He can help point out pitfalls specific to your project, business, or application area. For example, if you're building your first e-commerce site, you'll want someone who can tell you how to keep your customer's credit card information secure. Or if you want to set up an extranet application that allows your clients to check on the status of their jobs or payments online, you'll want to know how to keep their sensitive information private, while limiting the risk that other information will fall into the wrong hands.

3. **Don't commit to which features you want until you need to develop them.** This may sound like poor planning, but it's actually the opposite, because the good planning waits until the last responsible moment before committing. That gives you more time to gain the experience at each step to choose the most important features to add next. Who knows? That nifty little feature you were so sure you needed might turn out not to be so important after all. If so, you can always omit it. You can even cancel additional low-priority features at any time they start costing more than they're delivering in value, and you'll still have all the more important features, already deployed and being used by actual customers.

How do you determine which features are more important? That's a complex question, but there are basically two ways to measure importance. (1) You can look at the business value, which feature is worth the most to your business. (2) You can look at how much the feature helps the developers learn about the system and its technology. Features in category 2 are frequently not even "features" from the user's perspective, because they enhance the system's infrastructure or impart knowledge to the development team. But they are important in reducing risk, because lack of knowledge is the biggest risk that can derail of software project.

4. **Make sure the contract allows for changes, and make changes before committing, but not after.** Because we live in an uncertain world, you need a plan to deal with that uncertainty. Make sure you're working with a development team who will allow you to make changes to the plan, up to the point at which they begin actually working on your requests. Then you don't have to commit to a certain set of features until the development team begins working on them. But once you do commit, try not to change your mind, because that will throw the project into a tizzy and just slow things down. It will be as if you told your electrician you want to move the bathroom sink against the south wall, and then just as he begins laying pipe, you say, "Actually, I think the east wall would look better." And then, "No. Better make it the south wall." "No, the east wall." And so forth. As you can imagine, software development teams tend to react badly to that sort of thing.
5. **Provide software requirements in the form of objective test procedures.** Don't just say, "I want a feature that will do X." Rather, go step by step through a typical usage scenario. The user does A, and the system responds with B, and then the

user does C, and D happens. This is a little more work up front, but it will save you grief later on when you need to decide whether the system does what you asked. It also provides the developers with an objective standard they can use to make sure they're doing their job right. A good software engineer will ask you for a scenario (or a bunch of scenarios) like that, and he may even suggest alternatives that might make the system work better.

6. **Set objective acceptance criteria for each feature.** This goes hand in hand with the previous tip. If you've already thought through what the system is supposed to do, you can just check that it's doing that. However, you will probably find that for all your careful planning, the developers won't be able to make the system do *exactly* what you had envisioned. Be as flexible as you can without giving up your core requirements. Think through ahead of time what you *really* need and what you can get away with. Then talk to the developers about what they can do to give you the former.
7. **Ask the developers how long each feature will take to implement, and whatever they tell you, trust but verify.** Software developers are generally optimistic about their estimates, unless they have a process in place to measure how long they actually take and adjust their estimates. Even big outsourced development companies can be over-optimistic about their estimates, because once their client has invested months of funding into a project, it's pretty hard for them to pull out once it becomes clear that it'll cost twice as much and take twice as long to finish. That's the risky approach. You want to find a development team who will release in tiny iterations, no more than a month each. And if they tell you features X, Y, and Z will take a month, believe them, but measure whether it really takes a month or whether it takes 2.
8. **Make an "expert user" available to advise the developers.** Chances are, your developers only have a partial understanding of how you and other users will use their system. Good developers will always be coming back to you with questions about how the software should work, questions that you probably never even thought of before. That's because the developers are getting down into the nitty-gritty logic of the thing, as a mathematician would, and coming up with numerous edge-case scenarios. That's a good thing, because it's better to resolve these issues now, during development, than later, when one of your customers runs smack dab into one of those edge cases. So have an "expert user" on hand to

answer these questions, someone in your company who can represent the users of the system to the developers.

9. **Get regular status updates.** Real software engineers don't just retreat to a cubicle for a few months and then magically emerge carrying the product of genius. Rather, they are constantly watching, constantly evaluating, constantly asking questions, constantly making suggestions. Remember those frequent releases from tip #1 above? Have someone look at them. In between releases, meet or talk with your contract house. The developers working on your project should feel like an extension of your business. It's okay for you to insist on seeing intermediate progress.
10. **Expect the possible, but not the impossible.** Let's say your developers tell you it will take 2 weeks to develop features X, Y, and Z. And let's say in the past, it's taken up to twice as long as they've said it will, so you can expect up to 4 weeks before X, Y, and Z will actually be ready. And let's also say that you need X, Y, and Z in 1 week. What do you do? Well, you can look for a way to loosen the time constraint, to give yourself 4 weeks instead of 1. You can also talk to the developers and see what you can get in under a week that will meet your needs. Maybe you actually only need X and part of Y, and maybe that much is really fast and easy to develop. Maybe it was Z and the other part of Y that was the hard part that took multiple weeks to develop.

Whatever you do, resist the urge to "push back" at the developers to try to get everything faster, because you'll most likely be disappointed with the results. Remember that even big consulting houses can be over-optimistic about the project schedule, because clients love being told that the impossible will be done in no time flat. Don't believe it, and don't ask for it, because you don't want to have to depend on pie-in-the-sky estimates that can't come through. Rather, ask what your developer can accomplish and how fast, and then trust but verify. Most developers are happy to give honest estimates if they think they'll be listened to. Use their expertise to plan accurately.

11. **Be prepared to choose which features are more important, and which are not.** Very commonly, when I have two features, A and B, both of which "must be in next week's release," I can only get one or the other done. And I can't just frob off the work onto another developer on the team, because A and B both require my

specific expertise. Besides, all the other developers have their own list of things that “must be in next week’s release.” Of course, in truth, I don’t know that I can’t get both A and B done. If all goes well, I might be able to fit them both in, but I don’t want to bet my career on it. So I explain the situation and ask, “Which is more important, A or B?” And invariably, the client answers, “Well, both are.” I reply, “But if you *had to choose*, which would you rather have, and which would rather put off until the next release?” This is the tough choice that has to be made at least once in each project. I’ve never encountered a situation where there wasn’t an answer, nor where that answer meant the death of the project.

12. **Expect to fail in the short term, even if you’re optimistic in the long term.** When Microsoft introduced Windows, it was a toy. Then they came out with version 2, which was a joke. It wasn’t until Windows version 3 that the industry sat up and took notice. Now, it’s the de facto standard operating system for desktop computers. You’ve probably experienced this pattern over and over again in your career and your business, and you should expect it with your custom web project. This is especially true if you’re bringing in a new team of engineers to work on an already existing project. If it’s like most such projects, the new engineers will seem to be fumbling around it’s maze-like corridors and blind alleys at first. But a good developer will simplify and improve the software as he works on it, smoothing out its kinks and eliminating its hidden death traps, and his efforts will bear more and more fruit as he continues to work on the project.
13. **Don’t be afraid to pull the plug.** This is something software development contractor hates to hear, but it’s critical. If you’ve been following my advice so far, you’ve established a relationship with your developers, where you’re prioritizing their work in tiny bites, negotiating with them for realistic time schedules, and measuring how much each bit costs and how much it’s worth to you. You also probably started with a long wish list of custom features, but you have been putting in place the most important ones first, leaving others for later. At some point, you’ll find that all the most important features are already working and that the ones that are left aren’t actually worth what you’re paying for them. In economic terms, the marginal cost of each new feature exceeds its marginal benefit. What to do? Stop buying new features. Put the project in “maintenance mode,” which means you’ll only need someone to work on it to fix things that go wrong. Anyway, by that time, you’ll probably have a pressing new project that you’ll want to start work on.

Note that I didn't include items like "Always get a copy of the original data files for your project, in case the contractor bails on you" and "Make sure you have a backup of your web data files," both of which you should indeed do. But those are warnings that apply regardless of how much or little customization is involved. These 13 are *software development* concerns that apply when you need to develop custom features.

And not every development shop will be able to work within these bounds. These tips do come from many years of experience, failed projects, and successful ones. But there's also a certain amount of faith involved, because despite the years of experience, we still have scanty scientific data to prove which software-development practices are most effective. That's why some developers may not be used to working with the kind of process I've outlined here and may even refuse to. So you may need to shop around a little to find a developer who can. But at least now you know what questions to ask.

J. Timothy King is a software veteran, having been developing software since the 1980's and developing for the web since the web existed. His expertise is in quality development practices, including automated code testing, test-first development, refactoring, designing for maintainability, and working with legacy code. He writes about software development on his [programming articles site, JTSE.com](http://JTSE.com).