4 Things Not to Do with Your Internet Project

Your not-for-profit is launching, expanding, or rebuilding its Internet presence. Congratulations! Now what?

Our partners and key technical staff have more than 40 years combined experience working with not-for-profit organizations in the capacity of technologists and technical directors and have seen firsthand the many successes—and missteps—organizations make while planning and executing an Internet project. If you're looking to build, maintain, or rebuild your presence on the Internet, we've identified the four most common pitfalls to avoid. We hope that sharing these here will help you with your own projects.

1. They hire a programmer instead of building a team.

Many organizations who wish to bolster their online presence start by hiring a programmer. They expect—or hope—that this one person will be able to act as an in-house technical resource for the organization. The programmer comes on board and does what programmers do: they write code. This arrangement seems ideal at first. However, a fundamental issue begins to show itself—often after a surprisingly short period of time—namely, that most programmers are not skilled in technical architecture or in running projects.

Without proper technical architecture, the project will fail. Think about it this way. You want to build a house, but you hire an interior designer. What will you get? A basement that has been poured too shallow or a water heater that can't support the jetted tub in the bathroom. The bathroom fixtures might be top-of-the-line, but you'll be taking cold baths. And without a proper basement, you'll fail your building inspection. Technology works the same way; the site may look sound, but it may not work as expected nor be able to scale to a system that can service real users.

Additionally, programmers are usually not skilled at running projects or managing project budgets. Organizations tend to turn to their internal programmer and say, here is what we want to accomplish, how long do you think this will take you? They accept the programmer's answer, walk away, and at some point, they check in. Ninety-nine percent of the time, the project is off track and off budget. Why? Well, programmers—certainly the ones you want working on your projects—enjoy dropping into and solving tricky technical problems. That is where their focus lies, not on the management

Remember one other thing: **it is your team, regardless of which members are internal or external to your organization or work with you full- or part-time.**

of a project or the bookkeeping of a budget. Someone needs to watch their progress and switch their focus when necessary.

So how should you proceed?

The answer is, you need to create the right team of people, not simply find one technical person.

Start by looking internally for the resource who is familiar with the project and its business requirements. This is the person who will manage your project, including participating in budget reviews.

From there, look to fill out the team with technical staff. You will need an experienced software architect, someone who thinks about architecture and system design first, coding and languages second. This person should be able to offer recommendations for how your project can be set up and be senior enough to manage the programmer or programmers who will fill out your team.

You may be reading this and thinking, we have budget for a salary or two, not a whole team! Well, generally, design, programming and architecture are better outsourced to companies who specialize in that work, while other team members, such as product owners and project managers, tend to be internal resources. Keep in mind that unless your project is large, some of the technical team will be part-time contributors. Remember one other thing: it is your team, regardless of which members are internal or external to your organization or work with you full- or part-time. This mind-set will allow your team to operate cohesively and fluidly, without unnecessary segregations between staff members versus contractors.

2. They second-guess their choices and tweak the requirements or business processes.

I can't count the number of times an organization has worked through requirements with me or one of my teams only to come back later and say something akin to, "We talked about this with our new president of the board of directors and he wants to do a few things differently. We need to make some changes."

What happens next is one of two things: (1) The product team says "OK" and continues to work as if nothing happened; or (2) the product team says "No, that isn't possible within the current scope of the project," and they present a change order either asking for more money or itemizing requirements that won't be met due to the requested changes.

Businesses will naturally be happier with the first response initially. Their reaction to the second option may be more complex. Perhaps they say, "What we've asked for is not that big a deal, what's the problem?"

It's difficult to see from the outside, but changes are like ripples from a stone thrown into a pool of water. Even simple requests can affect many areas of a project.

It's difficult to see from the outside, but **changes are like ripples from a stone thrown into a pool of water. Even simple requests can affect many areas of a project**.

The technical team will have to determine the scope of the changes and identify how they will impact other areas of the codebase. Progress can be derailed as programmers will have to rework code they've already completed. **Continuous changes can turn a project into utter chaos because the goal turns into a moving target**—plus it is inevitable that things will start to get lost in the shuffle.

The first response, the team saying "OK" without any pushback, is similar to kicking the can down the road. As the project nears its supposed completion, production will lag further and further behind. By the end, either entire portions of the project won't have been completed, or they will have been rushed into place at the end. As a business, you'll feel immensely frustrated because you didn't have a say in what happened. Perhaps you wouldn't have asked to add international members to a previously US-only group—which seemingly only required a separate address form, international postal codes, and alternative currencies, but actually included addressing more stringent EU privacy standards, requiring stricter encryption choices, altering public-facing user profiles for EU versus US citizens, and reworking what kinds of personal data you need to delete when a EU citizen requests it versus a US citizen—if you'd known it would be at the expense of a different critical aspect of your project.

The second response, while initially more upsetting, allows you to continuously manage the scope of change. It provides the means for your team to make adjustments as necessary. If the change that your new president requested has greater consequences than you expected, you'll be able to discuss the ramifications of that change and weigh its value against its cost.

3. They build the product but do not maintain it.

Online products require constant maintenance. Software packages get old and need to be updated. Security requirements are constantly evolving. Browsers and mobile devices are continuously upgrading. Every modern online product is ultimately dependent on many other vendors and software packages. Needs change and features require alterations and enhancements to accommodate them.

Not-for-profit organizations tend to underestimate the cost of continuous change in two ways. First, they budget for the construction of their product but not for its ongoing maintenance and enhancements. Inevitably, they find themselves in situations where something goes wrong, or they need to make improvements, and there is no one (and no money) to fix it.

The second is, they keep pushing forward with new features without taking the time to address back-end needs or to upgrade underlying packages.

It takes a technical team significantly more time to work with outdated technology than it does to work with newer technology. Sometimes the issue is logistical—outdated hardware connectivity may become crushingly slow—but there are many other reasons why this is harder, from

Maintaining software is somewhat akin to maintaining a house. You wouldn't expect to purchase a house and never maintain or upgrade it again.

identifying what functionality was—and was not—available, to relearning how older frameworks work that have been updated several times since, or having to wade through outdated documentation (if they can even find it). Eventually, the technical team will run out of road with the outdated technology, and when you get to that point, you will be forced to upgrade.

So how should you proceed?

Ensure that your team provides you with understandable, realistic upgrade and maintenance plans.

It could be difficult to know if a plan is realistic without technical expertise, so the emphasis should be placed on determining if the plan is understandable. Some questions you can ask include:

- What is the plan predicated on—is it just anticipated software changes or does it take future growth into account?
- Are you given specific dates (as specific as possible) for when key elements of your infrastructure will require upgrading?
- Are you in a position to address security issues that may arise quickly?
- Do you expect that you will continue to add to the product, and if so, are you differentiating between "new features" and "maintenance of existing features" in your thinking?
- Are you accounting for hardware and hosting costs as well as personnel costs?
- Can you retain the team that you invested in to build your product? If not, are you accounting for the time it will take the existing team to fully document their work and the new resources to come up to speed with your product before becoming productive?

Maintaining software is somewhat akin to maintaining a house. You wouldn't expect to purchase a house and never maintain or upgrade it again. And while you may have as little expertise with online technology as construction, by asking the right questions and realistically considering your future needs, it is possible to understand what you might need to maintain your product.

4. They overengineer or bite off more than they can chew.

It takes vision—and guts—to build out your website or Internet product. You're investing in what you know your users need from you. Chances are you've spoken to quite a few of them and gathered feedback about what they want to see. You've looked closely at the niche you're wishing to fill versus what your competitors are doing. In the end, you are looking to build something because you know it will help your users.

The thing we've found is that visionaries can see the end result of their visions much clearer than the seemingly plodding, execution-oriented steps that need to occur to get there. They also generally have a blind spot: they can't gauge up-front how users will respond to the finished product. The actuality of real users in your system can, and will, dramatically change the initial vision of the project. You'll need to be prepared to gather that engagement information and adapt to it.

The actuality of real users in your system **can, and will, dramatically change the initial vision of the project.**

The best way to do this is to always build the product you need right now instead of the one you think you'll need in 1, 2, or 5 years.

This doesn't mean that you aren't planning for that—it simply means that when you are making decisions about which aspects of your product to build first, how expansive your technical environment should be, or how quickly you should push out functionality, ask yourself what the feedback cycle from your users will be, and work to speed it up. How can you get functionality into your users' hands—and then get feedback from them—more quickly?

We recommend the following steps as a process:

- Engage a subset of your users early in the process. (If you don't have the users yet, gather people who are outside your organization and unaware of your vision to help you test.) Analyze their reaction to prototypes or even designs of what you are planning to build. What works for them? What confuses them? What priorities can you determine from their reactions?
- Build the highest-priority items first, release them quickly, and then take the time to gauge usage, acceptance, and other internal benchmarks, such as any reporting or KPI metrics your board or donors might need.
- Build your infrastructure for the users you plan to have in the next 3 to 6 months. Don't build, or wait for, the grand infrastructure you think you'll need in 5 years. When you have that competent system architect on your team, she will build the system in a way that it can scale in the future. If and when your product takes off and you need to expand your infrastructure, your system will be ready for that step.

What we've seen is that business leaders focus readily on the second and third bullet points, while the first, engaging users (this step is sometimes called user testing), falls out of budget. This is a mistake: the value of user testing cannot be understated. Also, these steps are not one-time steps. As you continue to iteratively improve your product, they should be repeated to ensure that you continue to stay on track as you grow.

We hope you have found this document useful, and can more clearly see how your organization can successfully plan and execute Internet projects.

If you have any questions, please reach out to us at support@coat-rack.io or visit us at https://coat-rack.io. We're here to help.