

## New Technology, and Balancing the Fun with the Practical

by Nolan, May 2009

Following technology trends on the web is a dizzying pastime. New programming languages rise to popularity seemingly every month and the buzzword-laden web frameworks make their introductions and dance on stage for half a verse before bowing out to younger, fresher technologies. When deciding which pieces of new technology to build into your web presence, there are several things that a developer should consider and some simple guidelines to pull in new, fun technology.

Obviously, the first attribute that a technology is measured by is its **quality**. Simply, is the code behind it good and sturdy? If you lean on it too hard, will it break? If you sense [code smell](#), follow your gut and keep moving.

The next attribute I look for is **community**. You read about a new technology and it seems to be the silver bullet for your application. (We'll disregard for the sake of the example that there is [no silver bullet](#).) Reading further you find out that the technology is driven by one developer. The online mailing list gets a question once every two days, and the IRC channel is frequented by the lead developer and a fistful of other passersby just checking things out. That is no community; that's a household.



Django Community Sprint, Photo by [Nathan Berlor](#)

Now, almost every software project gets started with meager few people involved, and maybe this interesting new technology is in this stage of its life, but what if it's not? Maybe the lead developer hits the lottery and moves to the sunny French Riviera, never touching his MacBook Pro again except to order fine wines for his chateau. You now have a software project that has possibly no active developers and the project fades away to the great Internet Archive in the sky. If you had adopted this technology for a website, you are now riding with an effectively dead technology. This is not a good situation to be in when you need to hire someone that knows it to maintain your site, or to fix bugs that may pop up in the software.



The French Riviera, Photo by [Serge Melki](#)

With community, a software project has staying power. The more people involved, the higher chance it will be around longer.

Signs of good community:

- 1) A local user group
- 2) Corporate backing (not essential, but definitely nice)
- 3) Lots of blog posts and Twitter messages indicating that people are talking about the tech.
- 4) Active mailing lists, both for general help list and the developer list.
- 5) An IRC channel where you can pop in and find numerous people to help you with an issue.

All of these are not necessary to have a good community but the major software projects tend to have these in common.

My personal favorite trait in assessing a new hunk of technology is **documentation**. Well-written and kept-up documentation is something that makes most developers happy, but unfortunately writing documentation is not enjoyable as a task. If a software project has a fantastic demo, and then I click through to the documentation wiki and find more instances of "coming soon" than code samples, I usually step away.

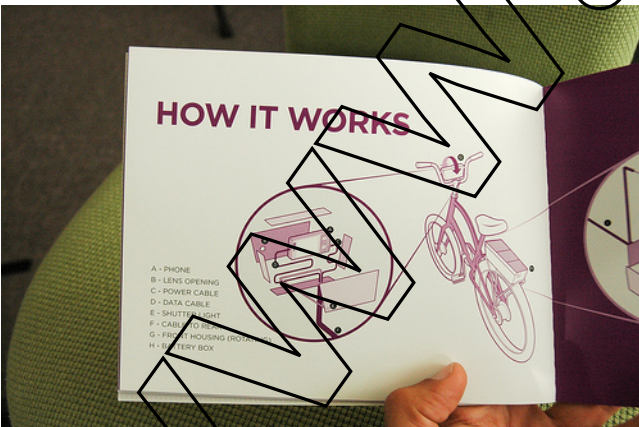


Photo by [Amit Gupta](#)

Lack of documentation and time to implement a technology are usually inversely related. Little documentation results in lots of tinkering and cursing, fumbling in the dark and looking for that magic combination of words to get the spell working. Software is painful enough;

building with it without proper guidance is torture. Some developers thrive on the poke-and-prod method of development, but when a client's money (and the developer's sanity) is on the line, a few breadcrumbs goes a long, long way.

Now that some simple guidelines on vetting technology has been laid down, how do you bring the technologies that look fun and promising, but may not be quite ripe, into the workflow? Easy. Start small and leave traces.

Starting small means don't build anything mission-critical on it. You definitely don't want to nab a big client and then decide to build their website on a funky schema-less graph database you read about on Reddit. I can guarantee you they don't want to be your testing ground. Now, you may grab another client whose needs may line up for the new and edgy technology. Let them know what you are doing and be honest about your experience with the tech. No one enjoys any surprises in budget or functionality further along.

Leaving traces means let your co-workers know what you did. On your dev team's wiki (if you don't have one, you should), document out why you chose the technology that you did and how you used it, with lots of links and code samples. In case you hit your numbers on the weekend PowerBall and retire early, you don't want to leave your comrades high and dry poking through your code when a bug pops up.

The web is a fun place, with lots of smart people doing cool things every day. Playing with new technology is why a lot of web developers do what they do, and bringing this into enterprise doesn't have to be a pipe dream. So, feel free to experiment, but just be responsible.

\*All photos licensed under the Creative Commons

www.pdflib.com