At *Symfony Live San Francisco 2012*, I gave a little talk. No, really. A *little* talk. **Seven minutes**. I'm not even sure I used all of it. That's not a lot of time, but I think I managed to at least provoke some thinking. At least I hope I did.

Hmm. How *do* you act like you care about your work, as a developer?

Community

In order to be an effective developer, you must take part in your community. Even if you are only consuming, and not creating or contributing, knowing your community is a must. It's how you keep up on your trade. How could you know what the prevailing best practices are if you don't have your finger on the pulse of your community? Trends shift and recommended modes of development are ever-changing.

Years ago the mere act of creating object-oriented code in PHP was still hitting its first real strides. Now, we are exploring things like aspect oriented programming (thanks to traits support in PHP 5.4) and are comfortably on board with decoupled objects being strung together by a dependency injection container.

How amazing is that?! It is a massive evolution in the collective code quality of the PHP community! Tracking the pulse of your community is an essential part of becoming a better developer.

Education

I am not talking about school here. I am talking about learning, upping your skills, bettering yourself. It is vitally important to continuously improve your skill set! Every day, the knowledge you knew yesterday gets another day older. In this industry we can't even accurately predict what we will be doing in ten years. The web as we know it today will be radically different. At least I hope it will!

Fabien Potencier said in the round table at Symfony Live San Francisco 2012 that it is *essential* for you to be able to look at your code from six months or a year ago and be nearly devastated by how bad it is. And he is right! The rate of change that Moore's law defines doesn't just apply to hardware. If you haven't had a year-over-year improvement to the point of being utterly **embarrassed** about your code from a year ago, seriously, go find another line of work. I hear that McDonald's is hiring.

It has been said that it takes 10,000 hours of doing something to become good at it. How much time have you invested in your craft? How much time have you invested in learning *how to learn* your craft?

Learning is an Investment You Make in Yourself

If you aren't setting aside a few hours a week (minimum) to invest in yourself, you are already behind. Short of maintaining a legacy platform, new skills and experiences are essential to maintaining your edge. Can't think of anything?

- Composer - Dependency management for your php
- Bower - Dependency management for your javascript and css
- PHPUnit - Testing! (more on that later)
- Try Ruby - A nice 15 minute tutorial on Ruby
- Khan Academy - Tired of programming? Learn something random.

Not everything you learn has to be programming related. It's the consistent **act** of learning that is essential. Being used to learning new things keeps you ready to learn new things. You'll never shy away from a challenge because you are unfamiliar with it. You'll see it as an opportunity to grow.

Learning To Learn

Learning to learn is a widely discussed topic. To be a great developer, you not only have to learn new things, you have to be able to learn new things quickly. Project requirements often change rapidly, and the faster you can adjust, the more successful you, and your team, will be.

By now you may be saying “Ok, I like this whole learning to learn business, but I just don’t have time!”

I will teach you to do the impossible: to make time.

Automation

Now here is where I practice what I preach. You know that learning stuff I just talked about? I do it. Every day. There are all sorts of things I haven’t done for 10,000 hours. Want to know one? Lets talk about automation. There are tons of tasks you do every day. I am willing to bet a vast majority of them can be automated in one way or another. I am going to show you two areas where automation can buy you time to reinvest in yourself.

Automated Testing

Automated testing includes anything that you can invoke via a single primary command. That means unit tests, functional tests, acceptance tests, and the like. Anything that can help you *verify* that your code is performing as it should. You know how good of a programmer you are. You are fairly confident in your ability to patch a bug. You trust yourself to not screw anything up. I know you feel like that. I feel like that! Or I did, anyway. I have since adopted a slightly different tact: trust, but verify.

Unit testing is more than checking for bugs. It can help you think through design, as well as help you spot obvious flaws in an implementation early in the design process. Functional tests help you ensure nothing you have changed in one file (or system) has affected another one. Even better, when used effectively and after you have created at least some basic level of coverage, automated testing gives you one treasure that almost nobody seems to mention: confidence.

A comprehensive test suite with both unit and functional test cases gives you confidence. You will know, for a fact and without a doubt, that every scenario you can think of has been checked. By running a single command. Repeatably. Any time you want. Night or day. With just a few short taps on the keyboard. Let me tell you: it is an amazing feeling. It lets you make changes – both large and small – with the greatest of ease. Worried you broke something? Run the tests. Want to see how close you are to finishing your current code refactoring project? **Run the tests!**

Automated Deployment

Now that we have our test suite, we can easily ensure that our code base is in working order. Let’s reduce another point of risk and incorporate automated deployment into our (not so little) bag of tricks.

At its core, automated deployment is about adopting a predictable method leading to one of two outcomes:

1. Your deployment is successful, and your updated site is live and ready to go
2. Your deployment failed, and the publicly accessible version of your site was never changed

That’s it! If everything went swimmingly, it puts the new code live, and if something failed, it doesn’t.

That being said, applying the right technique can take some work. For the time being, we will let you explore the right technique on your own. It can range from very simple to very complex. If you’re looking for a starting place, I would recommend Capistrano, and if you’re using Symfony, you should definitely take a look at Capifony. They (and tools like them) will make your life much simpler.

A nice side effect of automated deployment is that adopting a deployment strategy forces you to look at other pieces of the puzzle, such as database migrations. If you use Symfony and Doctrine, be sure to check out the DoctrineMigrationsBundle.

With automation covered, let's get down to the nitty gritty and uncover a deep dark secret of software development.

Your Job Is Not To Write Code

That's right! Your job is solving problems. More importantly, your job is solving problems in a way that makes sense to the person (or people) you are solving the problems for.

Sure, entry level programmers are little more than code monkeys. In fact, according to some scales, entry level programmers are lower than code monkeys, not even allowed to commit to production systems. Over time you go from intern, to code monkey, to problem solver.

Once you have hit "Problem Solver", it's time to start owning up.

Let's look at this the most practical way we can: no matter what you're building, you always have a user to think about. Are you a backend developer? Then the frontend developers are your users. Are you a frontend developer? Then the users of the app are your users. EVERY DEVELOPER has a user they are developing for, even if it's just themselves.

Your job **is not** to write code.

Users vs. Stakeholders

There is, however, one other thing you should be doing. Some developers do this consciously, some subconsciously, and some not at all. You (and your team) need to act as referee between the project's stakeholders (the people paying for, or benefiting from, the project) and the users (the people using the end result of the project). It is your team that has the expertise to know when to push back against stakeholders to benefit the users.

Remember that keeping the users in mind, most times, will also benefit the stakeholders, via better adoption rates and better user retention. If it is an internal application that you're building, sure, people will still use it either way, but do you want people to scowl at your app all day? Not all of the arguments you make will be accepted, but it is up to you to try.

Co-workers

And while we're talking about jobs, your co-workers have jobs, too. For everyone else's sake, let them do their jobs. Really. Everyone has a job function to fill, and I swear – cross my heart, hope to die – that their job function isn't solely to make your life miserable. I promise. They are there to *support* the **product**. Just. Like. You.

It never ceases to amaze me when developers use phrases like "deal with" in regards to how to handle a coworker. SURE. In *some* instances it is warranted. In *most* instances, you're just being whiny about someone making you do your job. Remember that your job is to solve users' problems… in a way that actually solves their problems. In a way that they can (and do!) understand.

Designers

Designers save your bacon every single day. A good designer's work will keep users on your site, a **great** designer's work will convey emotion, engage your users, and draw them in to read your content, and hopefully use your application.

What do designers do? Well, all right, I will tell you, but don't say I didn't warn you. Designers take advanced knowledge of user experience, emotional design, and information architecture to build a site that is almost able to be scientifically provably awesome. If you want to know more about that (and you should!) you could check out any of the following:

- Color Theory for Designers, Part 1: The Meaning of Color
- The Elements of Content Strategy
- Designing for Emotion

Project Managers

These folks are another shining example of under appreciated positions in the industry. Project managers aren't there to be your friends (though they often are), and they're not there to take the fall when you mess up. Project managers are there to *manage the project*. This means sussing out problems before they become road blocks, while making sure you stay on time, and on budget.

Is your project manager being a jerk? Consider their side of things for just a moment. Have you delivered late? Have you made the project incur extra expenses? Chances are, if you're on the receiving end of some tough love from your project manager, there's something **you** can do to help things.

Some extra reading to find out more about what your project manager's job actually is:

- Project Management 101
- Wikipedia: Project Management

Marketing / Sales

Yeah, I know. The sales guy just asked you to add feature X to the system. It's going to take work. It's going to be hard. "It's not needed!" you say. "It doesn't make sense!" you say. Well, guess what? It's not your job to figure that out. Ninety nine times out of one hundred, when marketing hands down a feature, they have a customer (or more than one) willing to pay for it. Know what that means? Paying customers? Oh, yeah! That's how the company you work for affords to *pay your* ***paycheck***.

Want to find out more about what marketing does? These links will get you started:

- Patrick McKenzie's Blog
- Forbes: What Does Marketing Do?
- KnowThis: What Marketers Do

We've covered the three main points of contention, but let's not leave anyone out!

Frontend Developers

These people should get some kudos too. If they're asking you to do something, usually it means what you've given them, while it may be workable, isn't optimal. They're trying to make the site load and run as quickly as possible. That means the fewer requests they have to make back to the server, the better. It also means that the faster the server responds, the happier they will be, because these people work with the designers to ***make their users love your code***.

If you're building things for frontend developers, remember they're working with one of the craziest languages we use regularly on the internet. JavaScript would re-scope your family if you gave it the chance. So when you're building a frontend developer's support infrastructure, be good to them. Export your data in common formats – JSON for consumers you control, XML for those you don't – and make your API interfaces RESTful.

Backend Developers

Now, frontend developers, you have to give your backend developers some props. They work hard to make sure you have what you need, when you need it. If you need something, ask them. Don't be confrontational, just say "Hey, I could really use X. Is there any way you could make that work?" It's all about how you approach them. Frontend developers, I think, tend to forget that backend developers may not have as much people exposure. Don't run to them saying "HEY GUYS I HAVE TO HAVE THIS RUNNING WHY DOESN'T IT WORK NOW?!?!?! ZOMG?!?!?" That will surely earn you a place on their bad side. They won't tell you directly, of course, but you may suddenly see your feature requests taking a lot longer to complete, or receiving a lot of push-back, if they are even deemed possible at all.

As much information as frontend developers, designers, and marketers need about people, backend developers need about raw "stuff". In my own experience, backend developers **require** the most amount of faith because most of what they do, by its very nature, isn't directly exposed to the world. That doesn't mean to let them slide on deadlines – they still need to deliver – but unless you understand the tests and can follow the software's development yourself, you may have a hard time verifying your backend developer's status on a project. Trust me, they wish they could show you all of the progress they have made, just as much as you do. This tweet by Jeff Carouth sums up backend development quite well:

> "Well...uh...all the work I've been doing is mostly invisible until it's needed, at which time it will be invaluable" -- Jeff Carouth

And about all of the coworker stuff

None of this is, of course, universal fact. It will have some give and take. Not everyone is *qualified* to do the job they were hired to. Interestingly enough, not everyone who is *qualified* to do a given job is **actually any good at it**, and an amazing number who *are* qualified will never be given enough freedom to **prove it**.

Let us speak for a moment on something else we developers commonly have trouble with – forming healthy working relationships with our coworkers. **No** that does not include leering at the pretty receptionist. I said **healthy**. Make friends and learn to make small talk. Not only will you feel more like part of the team – and not just to you, but to your co-workers as well – you never know when you might need someones help to grease the wheels for a special request.

Just remember, as excited as you get about code, they get about design, or helping keep things on track, or understanding what makes people want to buy things.

You are all a team. You are all working together for a common goal. If that is *not* the case, then it is time for you to leave. Period.

If you don't know what it feels like to have a good team backing you up – you don't have one. Trust me. They may be an excellent group of people, but if you don't ***feel*** the groove, it's not the right team for you.

In Closing

There is one last thing; it's a tiny little detail, and really more of a personal favor. Too many of us merely build to the specification. Too many of us claim 'good enough' when we're simply tired of working on something. Stop it.

Endeavor, in everything you do, to build something that the end users truly love to use.

A special thanks goes out to Jeremy Mikola and Baldur Rensch for giving me initial feedback, the entire amazing team at Point 5 Foundry who push me every day to become better at what I do than I was the day before, and of course, my loving wife April Mather who has somehow managed to survive my passion for this work.

Oh yeah, and thank *you* for reading.