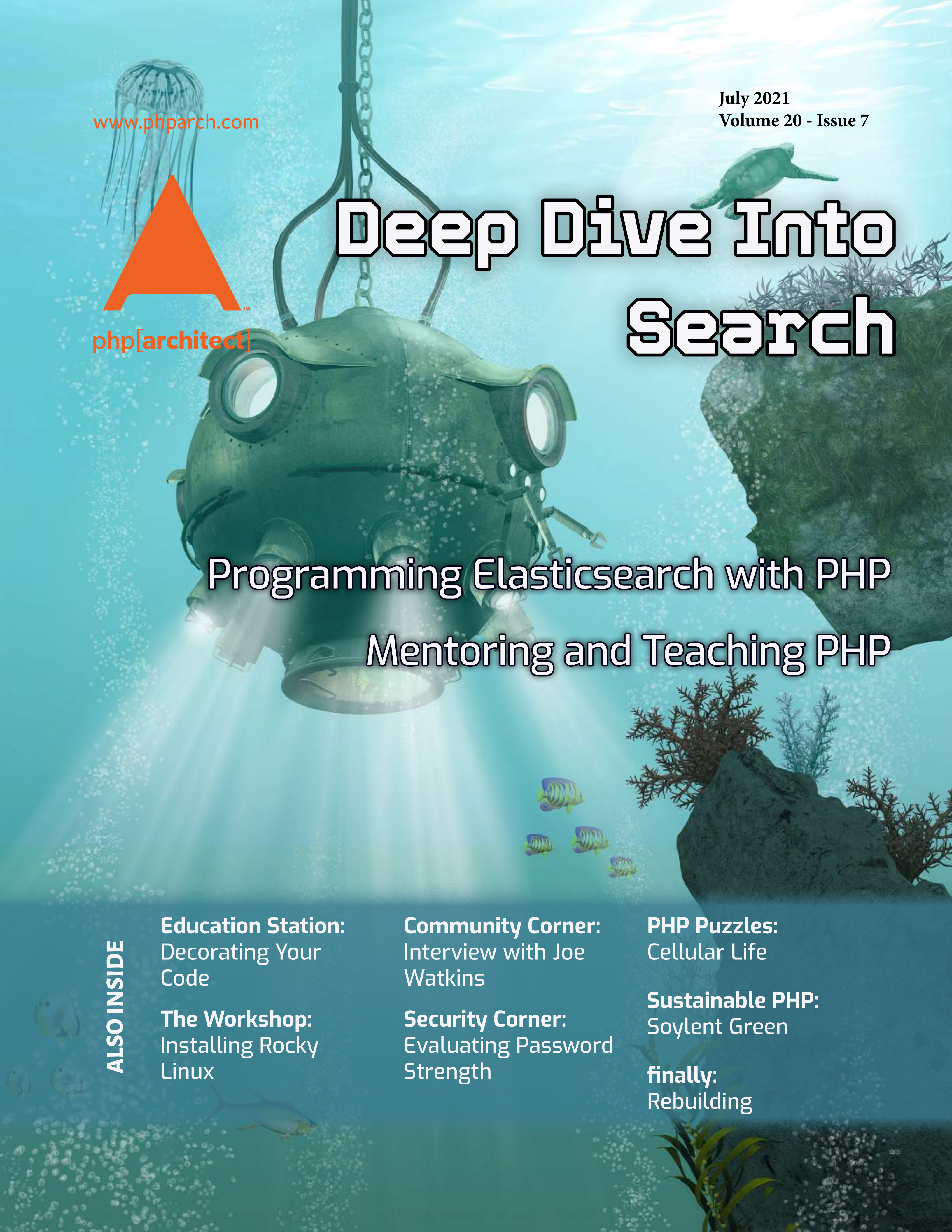

www.phparch.com

php[architect]

July 2021
Volume 20 - Issue 7

Deep Dive Into Search

Programming Elasticsearch with PHP

Mentoring and Teaching PHP

CLOUDWAYS

Moving Dreams Forward

Optimal PHP Hosting for Zero Downtime and Best Performance

Multiple performance tests show Cloudways improves **loading times for websites by 200%**! With innovative features like an **optimized stack**, advanced built-in caches, CloudwaysCDN, PHP 7.3 ready servers and so much more, Cloudways enables you to build apps with **unmatched performance** and **higher conversion rates.**

Promo: **PHPARCH**
20% off for 3 months

www.**cloudways**.com

Mentoring and Teaching PHP

Ken Marks

Hiring a new developer is a risky endeavor. What if they don't work out? What if they get bored? What if our organization's needs change? In this article, we'll discuss what it takes to foster a learning environment for your tech team, why you should encourage learning, what it takes to be a good mentor, and what you're looking for in a mentee. In particular, we'll look at resources and practices for encouraging new PHP developers to grow.

Why Does This Matter?

As an employer, hiring is challenging. How can I reduce the risk inherent to the process? Hiring decisions are often made based on the skill set you need right now, talent availability, and your hiring budget. A more proactive approach to hiring is to first think about the kind of environment your developers are working in. And by 'environment', I mean is your shop focused on performance and deadlines only, or is there also attention on your team's professional (organic?) growth?

Once you've made an honest assessment of your work environment, think about the environment you want to have. Fostering a learning environment for your team gives you more options when hiring for open positions. It gives you the chance to focus more on candidates who have a professional growth mindset and want to improve their craft. Individuals who can adjust to changing needs in the future and take on new challenges as they arise.

A learning environment incentivizes your more senior developers by giving them an opportunity to mentor newer developers. In an environment only focused on performance and deadlines, there is less incentive to mentor newer developers and more pressure to perform and meet deadlines. Eventually, your senior developers get burned out and start looking for other employment opportunities. A learning environment means you'll grow your junior developers into senior developers, and you'll likely see an improvement in employee retention. To be clear, I'm not at all advocating you shouldn't focus on meeting deadlines. Rather, factor in deadlines as part of fostering a learning environment for your team.

Ingredients of a Good Teacher/mentor

Now that we've talked about the benefits of a learning environment, what does it look like, and how do you build one? A good learning environment starts with knowing the ingredients of a good teacher/mentor and fostering those traits among your senior staff. To start, assess your leadership skills and style.

My father also spent his career as an engineer, later getting promoted into upper management. He once told me engineers make the worst managers because they tend to be very exacting and want you to do things their way. I didn't take him too seriously until I started assuming more management responsibility later in my engineering career. Once I began stepping away from my day-to-day coding duties, I was moving into uncharted waters. I did not feel nearly as confident managing as I did in my coding skills. I started noticing some of my peers in similar positions, and I observed that those who struggled to make this transition into management took on too much of the implementation responsibilities, delegated less, and micro-managed too much, which ended up stifling the growth of their team.

I'm not saying everyone will struggle to move from an individual contributor to a manager. However, it is good to take note of your confidence level and deal with issues that could hinder your effectiveness as a good leader. Make sure you are modeling the behaviors of a good mentor too!

Once you've taken an honest look at your current approach to leadership, a good teacher/mentor will express trust and confidence in their team. You probably have a good handle on who the more senior and junior members of your team are and their approximate skill set. If you don't, you should make that a priority. Knowing your team will help you genuinely express confidence and trust in your team. So, how do you do that? I'll be blunt! You do this by getting out of their way and letting them do their job. Let me unpack that for you. Of course, you must hold your team accountable for project goals, but don't stifle their creativity by micromanaging them. Resist the urge to force your developers into implementing a solution "the way you'd do it" because you have more experience than they do. Doing so is one of the quickest ways to break down the trust between you and your team. There are lots of ways to implement a set of requirements. Make sure you have a rigorous enough code review and testing process that won't allow shoddy solutions to get through—this is also a learning opportunity for your more junior developers. Here is a brief list of things you should be doing to foster a healthy learning environment among your developers.

Be willing to let your junior developers "twist in the wind" a bit

As an instructor, I often have a student ask me to solve a particular coding problem for them. The last thing I want to do is solve the problem for them and take away an opportunity to learn how to solve this problem independently. Often, I'll ask them leading questions and ask them if they've tried X, Y, or Z. As a manager, you will want to resist the urge constantly to jump in and solve one of your developers' problems and risk making them overly dependent on you. On the flip-side, don't just tell them to RTFM. Again, you're trying to instill confidence and trust in your team and guide them to a workable solution. You want to be approachable.

Push your developers to do their best, but at the same time, guard against them taking on too big of a problem for the given timeframe

I teach an honors project course that requires students to complete a project of their choice over a single semester (16 weeks). These projects usually involve a completely developed and hosted web application. I always encourage my students to do their absolute best. However, they typically want to bite off much more than they can accomplish in a single semester. Early in the semester, I require them to develop a project plan and talk about time commitments. Guard against your developers overcommitting to a development project by helping them evaluate ideas and proposed solutions and teaching them the art of task estimation. I'll talk about these next.

You're there to help your developers evaluate ideas and proposed solutions.

In the process of cultivating a learning environment for your development team, as you grow in your leadership skills, you'll also want your developers to propose ideas and solutions of their own. This will not only help their professional development, but it shows your active investment in their career and improves retention. As your developers propose solutions, help them objectively evaluate their ideas by teaching them the tools you use to weigh the merits and risks. Show them what you do to mitigate these risks. This usually involves taking a more active role in mentoring them and encouraging them to pursue proposals that seem more promising. You may work with them or have them work with someone more senior on the team to come up with an initial project plan or prototype.

Teach them the art of task estimation

In the honors project course I teach, I have students read some material on agile development practices. Then we walk through their project idea and break it down into features and user stories. From there, I have them break the user stories into tasks. Ideally, these tasks are measured in hourly increments, and no single task should take more than ½ a day. I find that tasks that take less than ½ a day are manageable and usually do not need to be broken down further. When a developer learns how to estimate tasks correctly, it dramatically affects and informs the solutions they propose in the future. You should be able to rely on your more senior developers to teach these skills to the junior developers. Going through this process can also highlight areas of uncertainty where they need to understand their requirements better.

Ingredients of a Good Student/Mentee

When you have a development organization that fosters a positive learning environment, you have more options for filling your open developer positions. But, what kind of candidate will be best suited to this environment?

The most successful students who complete their web development degree at our college have these traits in common:

They stay motivated and disciplined to learn despite obstacles when learning new material.

In my experience as an instructor, it doesn't matter so much whether a student struggled more to learn new programming concepts. Instead, it was whether they stayed motivated and were disciplined to spend the time working through the concepts and improving their programming craft in spite of difficulties they encountered. You won't often be able to suss this out from a potential candidate you're looking to hire. However, you can tell very quickly by the willingness a new hire shows to learn new things and how they handle constructive feedback when giving them suggestions for improvement. You'll want to look for this eagerness to learn from your potential new hires. Taking on an intern can be a great way to lower your risk for potential new hires. I'll talk about this below.

"When we get confused and frustrated, we push the boundaries of what we know (or can recall). We're stepping out of our comfort zone. "

They are comfortable with confusion

I know. This is a strange point. A former colleague of mine used to tell his students, "If you want to frustrate someone for a little while, give them someone else's program to run. If you want to frustrate someone for a lifetime, teach them how to program!" He meant that great programmers are puzzle solvers and love the challenge of solving them. I tell my students every semester, "don't panic when you don't know the answer to a problem right now. Get comfortable with the idea that you will be confused and stumped at times; that's okay. Use it as motivation to mull it over in your brain and come back to it as soon as you take a short break." Sometimes I have to remind them of it throughout the semester.

When they're frustrated, they're learning.

This trait is similar to the idea of getting comfortable with confusion because programmers are paid puzzle solvers. When we get confused and frustrated, we push the boundaries of what we know (or can recall). We're stepping out of our comfort zone. This is good. However, as a leader, you want to make sure your developers don't "spin their wheels" too long. This goes with allowing them to "twist in the wind' but not for too long.

As a programmer grows in experience (3–5 years), they need to learn when a solution is 'good enough for now'

I have noticed an interesting trend in some of my highest-performing students who take my advanced programming classes. Sometimes, these students like to hand in multiple iterations of a project assignment. In other words, they like to solve assignments in various ways. I often encourage this behavior because it gives these students more practice, and they'll always improve their programming craft. However, most projects in the business world cannot afford the time for every programmer to deliver a perfect solution every time. I think when a developer approaches the 3 to 5 year mark of their career, they need to learn when an implementation solution is good enough to meet the required specifications and to trust in the software development process set up by the organization—or know how to suggest improvements to the process). They need to learn the difference between feature additions, refactoring, and reducing technical debt. You should be able to rely on your senior developers to help propagate this knowledge.

What It Takes to Be a Good Programmer

I'm often asked what it takes to be a good programmer. After nine years of teaching students at the college level how to program and develop web applications, I'm still not sure how to answer this question, and it seems subjective. That said, I know what good program code looks like, so I can only surmise that if you can write good code, you're a good programmer. So, I'll couch my opinion in what good programmers do:

A good programmer writes well-formatted code (that conforms to some coding standard) that is easy to read and maintain, ALWAYS! A good programmer writes their code so that it is self-documenting and uses a good mix of comments where appropriate. "Self-documenting" means using verbose and descriptive variable names. A good programmer is always interested in improving their craft. They're always motivated to learn and improve constantly.

DIEGODEV GROUP

Web Apps • Mobile Apps • E-Commerce

Developers who care about the code they create, the communities they build, and the solutions they implement.

www.diegodev.com

Since good programmers are always motivated to learn, you should encourage them to spend a small percentage of their time learning some of these skills on the job and not expecting them to do this in their free time. Maybe encourage them to lead a "brown bag lunch" where they're willing to facilitate a walkthrough of some new technology. This format might take the form of a book some members of your development team collectively go through. Consider purchasing the books for the team, so they know the company is taking a tangible role in furthering their knowledge and education.

Good programmers also need to have learning opportunities to grow in their day-to-day tasks. Above, I mentioned it is not a good practice to force your implementation solutions on your developers. It can also be a learning opportunity for your more junior developers by helping them discover the process of coming up with their own solution as long as it meets the requirements. The bottom line is: look for learning opportunities for them to grow and balance this against short-term inefficiency. An excellent tool for this is pair programming. You don't have to do this all the time, but it can be advantageous for both your junior and senior staff to pair-program, especially when learning new skills. Aside from the obvious benefits to your junior team members, it helps your senior staff see the benefits of mentoring and promotes a positive learning environment.

Social Skills Matter

When I was interviewing candidates for development positions, it was a given that I was going to vet them for their technical skills. The thing that scares a lot of hiring managers, though, is, "if I hire this candidate, will they get along with the rest of my team?" So yeah, social skills do matter when adding a new person to the team. This aspect is essential to my fellow instructors and me. We devote a good chunk of time making sure our students have every opportunity to practice and learn good social skills. In my classes, I teach my students how to conduct code reviews and give constructive feedback. I also have my students present their final projects in front of the class. This talk includes both a feature overview and a code walkthrough of some of the more challenging parts. You want to model this in your organization. I found that when developers on my team had the opportunity to demonstrate their features technically, it helped instill a greater sense of confidence, not only in them but in the entire team.

So what are some of these social skills we should be looking for in a new developer we want to bring into our team? Highest on my list is being a good listener. I do think you can assess this skill during an interview. A good listener repeats back to the communicator the important points of what was communicated to them in their own words. This behavior is essential to model with your staff as it helps to remove ambiguity and promotes trust.

The Value of Debugging

Looping back to what it takes to be a good programmer, in my mind, what separates a good developer from a great developer is how fast they find and fix their bugs. I find improving this one skill translates into the most significant efficiency gain for developers. I spend most of my time with my students honing this skill by teaching them strategies to isolate and expose bugs. My students have ample opportunity to improve their debugging skills and must demonstrate they're capable before advancing in the development curriculum. One excellent interviewing practice is to let a prospective candidate shadow a couple of your more senior developers for a ½ day—a full day would be better if you can afford the time and resources. Having a candidate demonstrate their skills by fixing a few minor defects is a great way to model and observe their approach to debugging code.

As developers grow in their debugging skills, they'll quickly need to transition from fixing symptoms to fixing root causes. It is a standard practice to have newly hired developers fix defects for several months before transitioning them to the development of new features. One apparent reason is to get them up to speed on the codebase. However, with guidance from your more senior developers, they can be mentored to look deeper for the root causes of defects. An excellent place to start is to have them start looking at the design history file (hopefully, you have one of these) to understand some of the design decisions.

Promoting Learning in the Workplace

Based on the discussion above, let me give you a small punch-list of what you can put together to promote a positive learning environment among your development team:

Have a Training Budget

If you don't already have one, create a training budget. It doesn't need to be very large, and you should be open to soliciting ideas from your more senior developers. A quick Google search will give you plenty of ideas on how to put this together. More importantly, getting your team's input on this will serve you well.

Brown Bag Lunches

I mentioned this above. Encourage your staff to hold brown-bag lunches by purchasing the books for them. If you can, consider paying for lunch or at least provide snacks or drinks as further incentive for them to attend and participate. Include this in your training budget.

Host or Ssupport a User Group

If there is a user group or Meetup in your area, think about hosting or supporting them at your facilities. Meetups are always looking for venues with plenty of space and good access to the internet. Moreso, it is a great way for your company to build a relationship with the local development community. It's also good advertising for your company when you're looking to fill positions. If you don't have a user group or Meetup in your area, start one!

Take On an Intern from Your Local College

One of the best ways to start building a positive learning environment is to consider taking on a paid intern from your local college. Effective internships usually last 6 to 9 months and work best if you assign one senior developer to be their mentor. Make sure you have a developer who is willing to do this. Many senior developers would like to have the opportunity to mentor a junior developer, and I don't think it's hard to build a culture that encourages this. There are several advantages to taking on an intern:

1. As mentioned, it helps to promote a learning environment among your development team.
2. It helps to build your organization's reputation for organically growing your development staff. Don't underestimate the power this has to make your organization a number one choice when candidates are looking to apply for a position with you.
3. Many colleges run a career day for their students every semester, and space can be limited, so not every employer can participate. By taking on an intern from your local college, you can be reasonably certain you will be able to participate in their career fairs.

Work with an Instructor for Group Projects / Presentations / Speakers

Instructors are always looking for professional developers to come into their class, give a presentation, talk on a particular technology, lead a tutorial, or do a group project. Students are very curious as to what the expectations are on the job. This is a great opportunity for your developers to talk about their work environment and form a relationship with instructors and their students.

Tools They Should Learn (eventually)

Whenever I talk to hiring managers in the area, they inevitably ask me, "what tools are you teaching your students?" Different managers have a list of what they want my students to be trained in, which varies from manager to manager. But we all have a consensus on what programming languages and skills a new hire needs to succeed as a developer. Because I spend most of my time teaching programming, development, and debugging skills, I don't have as much time as I would like to train my students in all the tools I want them to know. That said, I do expose them to a few tools I think they should be familiar with:

PSRs

Following a set of coding standards is vital to ensure code is readable and maintainable. I require my students to conform to the PSR-1 basic coding standards in my PHP Web development courses, and I point them to https://www.php-fig.org/psr/. We spend more time reading code instead of writing it, so this practice allows a development team to work more efficiently.

PHPUnit

In one of my PHP courses, I show my students how to use PHPUnit to teach them Test-Driven Development (TDD) and automated testing techniques. TDD is an excellent tool for new developers to learn. It puts them in the mindset of developing code from the perspective of what can go wrong and increasing the maintainability by making it easier to debug. Teaching automated unit testing helps students to see the power of TDD in speeding up the testing cycle and encourages them to write more robust code.

Composer, Packagist, and Third-Party Libraries

In both of my PHP courses, I demonstrate how to install Composer and what a Composer package looks like. I show them how to use Guzzle to create a PHP web service. Starting this fall, I'll show them how to use Packagist to browse for other libraries they might be interested in using for their final project.

Frameworks

Currently, I don't teach students how to use a PHP framework, but I should. I plan on creating a three-week unit on frameworks in my Advanced PHP class. More than likely, I'll use Laravel but make sure to cover the basic ingredients you want out of a framework, namely routing, security, and MVC development. It's easy to go down rabbit holes when teaching various features of frameworks (like templating). Still, I think it would be beneficial to have a working knowledge of the core features a framework provides. Of course, your choice of frameworks should reflect the one(s) you use in your projects.

Good Community Resources

I always encourage my students to use widely available resources to supplement their knowledge. For example, I constantly refer to the PHP manual[1], and I expect them to become familiar with it. However, to be a well-rounded developer, I also encourage my students to participate in our local Meetup (Full Stack Madison) and sign up for web conferences, especially those that give good student discounts. Part of facilitating a positive learning environment for your team includes encouraging them to engage in the wider development community around them. Encourage your team to attend web conferences, especially the local and regional ones around you.

Here is a list of resources I have my students use to help them become better programmers:

- The PHP Manual, https://www.php.net
- PHP The Right Way, https://phptherightway.com
- LinkedIn Learning, https://www.linkedin.com/learning/
- The Grumpy Programmer's Guide to Testing PHP Applications
- The OWASP Top 10 and Cheat Sheets
    - OWASP Top 10, https://owasp.org/www-project-top-ten/
    - OWASP Cheat Sheets, https://cheatsheetseries.owasp.org

Final Thoughts

As both a manager and an instructor, I have found developers want to feel valued and have the opportunity to grow their skills. If either of these is missing, they will go looking to meet these needs elsewhere sooner or later. In creating an environment where your developers feel valued and have an opportunity to grow, you will be helping them to meet both of these goals. You will also be motivating them to be the best they can be in their craft. I hope you are motivated and energized to embark on the process of building a more positive learning environment for your development team.

Ken Marks has been working in his dream job as a Programming Instructor at Madison College in Madison, Wisconsin, teaching PHP web development using MySQL since 2012. Prior to teaching, Ken worked as a software engineer for more than 20 years, mainly developing medical device software. Ken is actively involved in the PHP community, speaking and teaching at conferences. @FlibertiGiblets

Related Reading

- *Late to the Party, but Nailing It! A Journey into Pair Programming* by Jennifer Schrader, February 2020. https://phpa.me/journey-pair-programming
- *History and Computing: Mastering the Craft* by Edward Barnard, January 2020. https://phpa.me/history-jan-2020
- *Six Things I Learned as a Lead Developer* by Gabor "Mefi" Nadai, January 2021. https://phpa.me/six-lead-developer

1 *PHP manual: https://php.net*