www.phparch.com

March 2018
Volume 17 - Issue 3

php[architect]

Long Running PHP

PHP Daemons and Long-Running Processes

Evolving PHP

Containers Are a Pile of Lies! Part Two

Hands on With Accessibility

Sample

ALSO INSIDE

Artisanal:
Illuminating Lumen

The Dev Lead Trenches:
Reviewing Code

Community Corner:
Lessons Learned Running a PHP User Group

Security Corner:
Signed Commits With Git

Education Station:
Twitter for PHP Development

finally{}:
How Many Tools?

Free Sample Article

nexcess™

beyond hosting.

We're hiring PHP developers

Sample

15 years of experience with

PHP Application Hosting

SUPPORT FOR *php7* SINCE DAY ONE

Contact **careers@nexcess.net** for more information.

PHP[TEK] 2018

The Premier PHP Conference
13th Annual Edition

MAY 31ST – JUNE 1ST

Downtown Atlanta

Full schedule announced!
Talks from experts at MailChimp, Salesforce, Etsy, Google, Oracle, Pantheon and many more!

sample

tek.phparch.com

Features

Columns

Sample

Editor-in-Chief: Oscar Merida

Editor: Kara Ferguson

Subscriptions
Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Advertising
To learn about advertising and receive the full prospectus, contact us at ads@phparch.com today!

Managing Partners
Kevin Bruce, Oscar Merida, Sandy Smith

php[**architect**] is published twelve times a year by:
musketeers.me, LLC
201 Adams Avenue
Alexandria, VA 22301, USA

Although all possible care has been placed in assuring the accuracy of the contents of this magazine, including all associated source code, listings and figures, the publisher assumes no responsibilities with regards of use of the information contained herein or in all associated material.

php[**architect**], php[**a**], the php[**architect**] logo, musketeers.me, LLC and the musketeers.me, LLC logo are trademarks of musketeers.me, LLC.

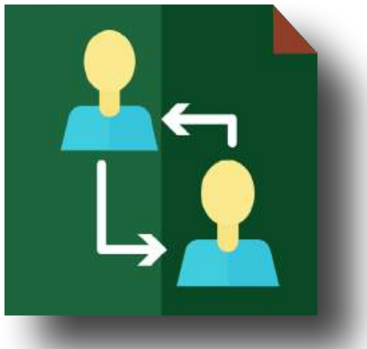
Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

Copyright © 2018—musketeers.me, LLC
All Rights Reserved

Reviewing Code

Chris Tankersley



Code reviews are one of the best ways to help a team ensure they're writing the best code possible. In all of the jobs where we have done peer-lead code reviews, we have caught more bugs and had better discussions about code than in places or times where we just hammer code through the approval process. I know, I know; we all write beautiful, bug-free code, so why go through the hassle of a code review?

Think of code reviews as analogous to test-driven development. In TDD, we write tests so we can confidently say we didn't break anything, and provide an additional layer of documentation for how we expect code to work. The computer ends up being our second set of eyes constantly watching for regressions in our code. If you use TDD, you know how nice it is to refactor some code, run tests, and know whether or not your refactoring worked or even made sense.

TDD also has a side effect of making you think about the architecture of your code. You will spend more time designing and laying out classes and structure than blindly coding until it works. This leads to more maintainable and cleaner code.

Code reviews provide a second set of eyes looking at the architecture and intent of code you write. If I am working on an issue, I may make some assumptions about how the system works, what users may or may not want to put up with, or just get too familiar with the code to notice things that need to be changed. Having another person look at the code can expose logic bugs or structural issues a computer just cannot see.

Performing code reviews can give you a better view of parts of code you do not generally work on. I review Python code from many of my coworkers quite a bit, and it helps me understand some of the changes they are making on their side of the application. I can better anticipate when we need to make changes on the PHP side of the application based on the code they are working on, and we can have better discussions about the direction of the software.

Code reviews will slow down how quickly code makes it into the mainline portion of your software, but I find the benefits—maintainable code understood by more than one person—far outweigh the downsides.

Code Review Tools

Most source code hosting systems provide a mechanism for code reviews. I think the built-in tools for GitHub and Gitlab work fairly well, and I more than likely already have other tools wired into these systems helping me with code management, like Jenkin's Pipeline system to handling automated testing. These default code review tools are usually more than capable of doing what I need, but there are some other options out there as well.

While I outline how to work with GitHub and GitLab in this article, I highly suggest using tools which integrate directly with your workflow. For example, if you are using the Atlassian ecosystem you will want to look at their Crucible code review tool. Any tool that integrates deeply into your existing software stack should provide you with a better overall experience than trying to bolt on random tools.

GitHub/GitHub Enterprise

Setup

Setting up a branch for code reviews is fairly simple, if somewhat hidden. By default, any PR can have comments on it and only collaborators can merge a PR. But you can go a step further and enforce that reviews are done. This gives you the added benefit of being able to control when something is going to get merged and make sure someone has looked at it and given it a once over.

To enable reviews, go into the **Settings** for your project. On the left-hand menu click on **Branches**. In here you'll find a section called *Protected Branches*. Here we can turn on the code review enforcement.

Choose a branch from the drop-down, and Github will forward you to a configuration screen (see Figure 1). The first thing you will want to do is check the **Protect This Branch** option. This will open up the rest of the configuration settings. You can now check **Require pull request views before merging**. This is the minimal amount you have to do, so you can click **Save Changes** at the bottom. Now any PRs made against the selected branch will have to have a code review applied.

If you want, you can also enforce other rules. I generally also turn on **Dismiss stale pull request approvals when new commits are pushed**. This makes it so if I approve a PR, but then someone pushes more code to that branch before I merge it, it revokes the approval. This is a good idea (and I think it should be on by default) since you do not want someone to approve a PR then slide some additional code in at the last minute.

I do not particularly like the **Require review from Code Owner** option as I do not think it completely necessary for someone to be a gatekeeper for a feature or a set of code. That leads to

Sample

siloing of information and tribal knowledge which might not get passed on. If someone wants to review a block of code they are not hugely familiar with, they can ask for help or converse with the "code owner" about particulars. I much prefer everyone being able to look at, modify, and critique any part of a codebase.

Reviewing PRs

Once the reviews are configured, we need a PR that is trying to merge into your newly protected branch. When creating a new PR, you can assign a reviewer then, or you can assign a reviewer later on.

When a new PR is generated against a protected branch, just above the comment box on the "Conversation" tab will be a big red X and messages stating the a review is required, and that the PR cannot be merged until it has at least one approved review.

Anyone can perform a review on a PR by going to the "Files Changed" tab of the PR and clicking "Review Changes." You can provide overall feedback and give an approval or rejection of the PR. You can also go down and commit on specific lines of code to start a review. At the end of your review you will need to either approve or reject the PR.

If you have general comments, you can leave them without needing to do a formal review. I do not do actual reviews until I am completely ready to either reject or approve a PR, since GitHub gets a bit funky about dismissing rejected reviews with no code commits. A good example is just asking a question about a line of code for clarification but then rejecting the PR overall. GitHub does not like to allow you to dismiss that rejection since no code was changed.

Sample

GitLab

Setup

I could not find any formal setup for enforcing code reviews in GitLab. The discussion tools are available right from the time a PR is generated, much like the general GitHub comment tools. The big difference is there is no way to prevent merging a PR until a code review has been done. This is not the end of the world, however, it may require you to do a bit more policing to make sure PRs are not going in before they have been reviewed.

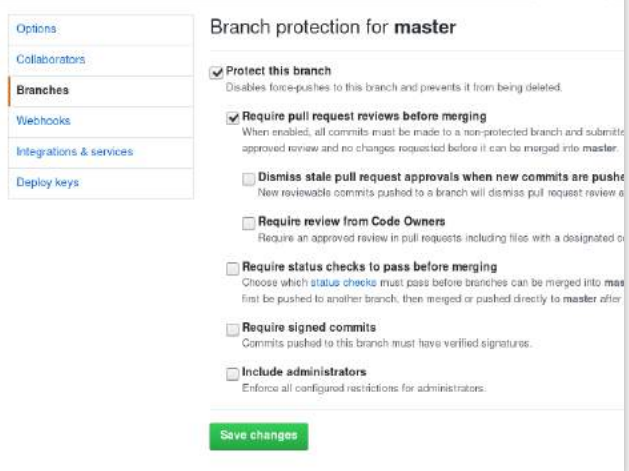
> *Merge request approvals[1] are available in the Enterprise edition of GitLab but not the Community Edition.*

Reviewing PRs

Since the community edition of GitLab is more freeform than GitHub, there is more of an onus on the developers to follow a workflow than having the tools handle workflow for you.

The first step is to submit a PR and assign it to someone to

Figure 1


Branch protection for **master**

- Options
- Collaborators
- Branches
- Webhooks
- Integrations & services
- Deploy keys

[✓] **Protect this branch**
Disables force-pushes to this branch and prevents it from being deleted.

[✓] **Require pull request reviews before merging**
When enabled, all commits must be made to a non-protected branch and submitte
approved review and no changes requested before it can be merged into **master**.

[ ] **Dismiss stale pull request approvals when new commits are pushe**
New reviewable commits pushed to a branch will dismiss pull request review a

[ ] **Require review from Code Owners**
Require an approved review in pull requests including files with a designated c

[ ] **Require status checks to pass before merging**
Choose which status checks must pass before branches can be merged into **mas**
first be pushed to another branch, then merged or pushed directly to **master** after

[ ] **Require signed commits**
Commits pushed to this branch must have verified signatures.

[ ] **Include administrators**
Enforce all configured restrictions for administrators.

review. This reviewer will go through the code and comment on individual lines. In GitLab this will generate *Discussions*, and all of these discussions should be resolved by the time the PR is ready to be merged. The first task will be for the main reviewer to go through the code, comment on any lines or changes needing clarification, and then pass it back to the original developer for changes, clarification, and/or arguing.

When the original owner gets the PR back, they can respond to any of the discussions that have been started. This response can be in the form of code changes or further discussion. Each discussion will have a **Resolve Discussion** button that can close out a discussion.

Once all of the discussions have been resolved, the PR can be merged.

Keep in mind a PR can actually be merged at any time, as active and open discussions will not stop a PR from being merged.

Code Review in Practice

This is technology, so there are some practices you can follow to make sure that code review goes as smoothly, no matter which side of the review process you're on.

Don't Punish

First and foremost, code reviews are not the time punish or call out developers. The entire point of a code review is to help each other write the best code possible, and demeaning comments or attacks in a code review are a no-go. This makes people not want to go through the review process for fear of being singled out. You should also watch your tone during the code review. Because you don't have body language and tone, text makes it easier to misinterpret comments as attacks; try and be helpful and make sure comments do not seem accusatory.

Ask Questions

You should ask questions about anything you are unsure of. Things that are clear to me as the original developer might not be clear to the reviewer; questions are a good thing. They can

1 *Merge request approvals: http://phpa.me/gitlab-merge-approval*

show potential clarity issues with the code which can lead to maintenance problems down the line. In the worst case scenario, a question might lead to a code change, but it might also lead to a simple response and explanation.

Have Clear Intentions

In either questions or recommendations, be clear about your intentions. Vague comments can just muddy the waters, especially if changes need to be made. Make it clear what the issue is and how you think it can be resolved. In the same vein, the original author might have a different way of fixing the issue. Compromise is a big deal, and we are all adults. Getting code accepted is not a win/lose situation. Work on a common fix.

No One “Owns” Code

Avoid the concept of “code ownership.” I mention it a bit above, but when someone takes ownership of code, it leads to rougher peer reviews. By considering code “mine,” you will be much more guarded against suggestions and changes than you might be about someone else’s code. Even if another developer built a large chunk of the code you are working on, we all are working on the same project. Feel free to consult or even have the original author perform the review, but neither side should assume just because someone wrote the code originally that it is in some way “holy” or “perfect.”

Provide a Full Review

When you are reviewing code, make sure you are providing a review of all of the code that you can, not just a single class or file. If you are not, just leave comments or questions. Nothing is more frustrating than resolving a bunch of comments only to have more appear just because the original reviewer did not look at all of the code. Each review should be a full review of all of the code.

Remember to not only look at the syntax of the code, but also the architecture of the code. Provide feedback on different tools or ways of performing the same actions. Is a class doing too much? Suggest breaking it into smaller classes, or suggest different libraries to use.

GitLab has a great article on performing code reviews[2]. I highly suggest reading through it in addition to my tips here.

Now Start Reviewing Code

I hope all of this helps convince you code reviews are a good idea, and that you can begin implementing them in either your open source contributions or at your place of work. Code reviews can be as simple as looking over code before it goes out, without any specialized set of tools. If you have dedicated tools, it can be more more controlled and be another gate keeper to making sure the best possible code is going out.

2 *performing code reviews:* *http://phpa.me/gitlab-code-review*

Sample

Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. He works for InQuest, a network security companyin Washington, DC, but lives in Northwest Ohio. He has worked with many different frameworks and languages but spends most of his day working in PHP and Python. He is the author of Docker for Developers and works with companies and developers for integrating containers into their workflows. @dragonmantank

php[podcast]

Hosted by Eric van Johnson and John Congdon, each episode of our new podcast compliments the theme of each magazine issue with interviews with contributors, thoughts from the Editor-in-chief, and more.

Available in iTunes, Google Play, and via RSS.

Listen Today
https://phparch.com/podcast-subscribe

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[**architect**]

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe

Sample