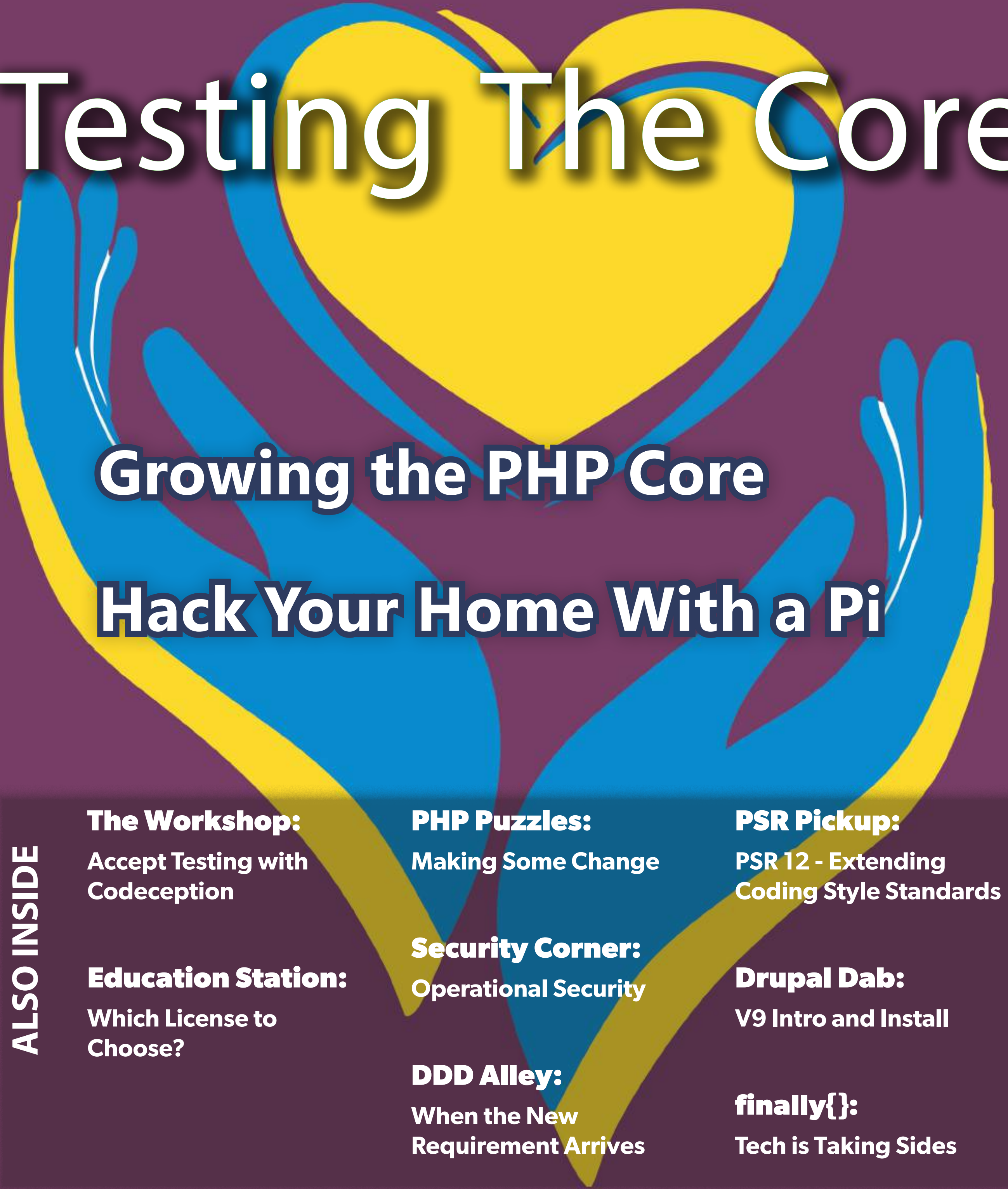
www.phparch.com

April 2022
Volume 21 - Issue 4

php[architect]

The Magazine For PHP Professionals



Testing The Core

Growing the PHP Core

Hack Your Home With a Pi

ALSO INSIDE

Which License to Choose?

Chris Tankersley

Licensing for software, whether it is open source or not, is an integral part of releasing software. The commercialization of software has made it necessary for developers to be explicit in how users or other developers consume their software. Unfortunately, the topic of licensing is not as straightforward as many developers would like it to be.

As a quick refresher, licenses are the legal terms that an end-user must abide by to use the software legally. This includes installing, using, or integrating the software in other pieces of software. Common things the license covers are personal versus commercial use, how and where software may be installed, and whether or not the end-user may modify the software.

There is much more to licensing than just "is this software open source or not?" If you are releasing software, you can choose from hundreds of available licenses. Why are there so many, and what are the differences between them? What you choose affects how users can interact with your software.

As developers, we need to be keenly aware of the licenses that we use in our own software. Licenses differ in the liberties granted to a developer. And different licenses can have very serious legal repercussions for a company. Have you ever worked for a company with a strict "No GPL[1] Software" policy? There is a very real reason for that.

While software can be used anywhere in the world, my experience and view for this article will be United States-focused. I am not a lawyer. If you are unsure how something may work in your country, I would consult with a lawyer familiar with your local copyright laws. Consider the following descriptions a view on the intentions of the various licenses rather than hard legal advice.

No License

Have you ever come across a repository that you would love to use, but there is no license file? Congratulations, you have come across a library or project where you legally have no usage rights. You should avoid this project at all costs.

A core component of any license is the rights and rules around using a particular piece of software. If there is no license, the copyright holder holds all the cards. They have not granted you any usage rights, and they have not granted you access to the source code, even if you are looking at the code on GitHub or another service. They may come after you legally for using their intellectual property without permission.

While I would love to be altruistic about this, this is the world we live in, thanks to modern copyright laws. No license means no rights, even basic usage rights. The project is just taking advantage of lax policy policing by code repository providers.

Public Domain

The Public Domain[2] is not so much a license in-and-of-itself, but rather a declaration of the abandonment of copyright on a work. In the United States, this declaration is known as "dedicating." A work dedicated to the Public Domain is usually identified with some accompanying text of "This work is dedicated to the public domain." Once a work is dedicated to the public domain, the work is no longer owned by any entity and free for anyone to use.

The Public Domain is incredibly problematic from a legal standpoint. The first problem is that, like licensing, dedication to the Public Domain must be declared. Many countries, including the United States, assign copyright automatically to the author. There is no legal authority one is required to go through to establish copyright (though there are legal steps one can do to help protect and declare the copyright).

Nothing is automatically entered into the Public Domain except under a few conditions. A work enters the Public Domains by being dedicated by the original author as mentioned above, or if copyright expires. Copyright can expire either naturally or if the author does not file for an extension. Once copyright is removed, a work enters the Public Domain.

While this sounds straightforward, copyright is a complicated thing. The rules differ for people versus corporate entities. Copyright is transferrable, and if this is not meticulously documented, copyright ownership can get cloudy. Authors of works may or may not have done so "for hire" (how most software is developed). Corporate acquisitions and breakups can make it hard to know who owns what copyright. Shifting rules of the length of copyright can make it hard to know if a work is due to automatically age out to Public Domain status.

Assuming copyright ownership is actually known, there is no legal definition of what "Public Domain" is. There are rules for copyright as part of the Berne Convention of 1988, but countries can still enact their own rules. For example, The US does not require dedication for works before 1988 and

1 *GPL: https://www.gnu.org/licenses/gpl-3.0.en.html*

2 *The Public Domain: https://fairuse.stanford.edu/overview/public-domain/welcome*

declared anything before 1927 as Public Domain (with exceptions). Copyright is also handled country-by-country so that a work could be Public Domain in one country but not another. This presents a second hurdle for anyone wanting to use Public Domain software.

Even scarier? The copyright holder could potentially rescind Public Domain status[3] with the way US copyright works. Now, all of a sudden, a library your company uses is no longer Public Domain. What do you do? Since there was no license agreement, it is unknown, both theoretically and legally, what would happen if such a thing were to occur.

At the end of the day, much like with no license, Public Domain licensed software is a minefield and should be avoided.

Public Domain-like License

What happens when you want the legal requirements of a license but the freedom of Public Domain? You get a variety of Public Domain-like licenses that effectively say, “I do not care what you do with this software, and I am putting that in writing as the copyright holder.” There are a handful of licenses available that fall under this category, but not all of these licenses are considered “Open Source.”

The Creative Commons is probably the most popular suite of licenses from this type of software license. The Creative Commons is designed as a variety of licenses for authors to use to better control how their works are used. However, it is not recommended they be used for software[4]. Creative Commons 0[5] is the closest to a Public Domain declaration you can get while still providing legal text.

Another popular license is the Unlicense[6], which includes anti-copyright language to make it more applicable around the world. It includes an official copyright waiver as well as a “no warranty” statement, which is an important declaration missing from many of the Public Domain-like licenses. In fact, the Unlicense is considered open-source compatible, unlike other licenses in this category.

While there are a handful of Public Domain-like licenses, many are legally dubious. One of the more famous examples is the “Don’t be a Dick[7]” license. On the surface, it looks like something akin to the Unlicense, but the main focus of the license is you are granted rights if you “aren’t a dick.” Unfortunately, there is no legal definition for what this means, and the license even mentions that the few examples given are not exhaustive. At any point, the copyright holder can just decide a user has broken the license based on whatever behavior the author deems is “dickish.”

3 *rescind Public Domain status:* https://www.techdirt.com/?p=83508
4 *software:* https://phpa.me/creativecommons-can-i
5 *Creative Commons 0:* https://creativecommons.org/?p=12354
6 *Unlicense:* https://en.wikipedia.org/wiki/Unlicense
7 *Don’t be a Dick:* https://dbad-license.org

It is my opinion that short of a Public Domain-like license approved by the Open Source Initiative, you should avoid these types of licenses.

Proprietary License

A proprietary license is essentially going to be any license that is unique to an individual piece of software. For example, when you install a piece of software like Microsoft Office, you agree to what they call an “End User License Agreement.” This agreement contains information that you might expect—it details how you may install the software, how usage is granted, and all kinds of other legal stuff.

While a company may copy-and-paste much of the text between their EULAs, each software’s license governs just that piece of software. The Windows EULA does not cover Microsoft Office, and the Windows 10 EULA does not cover Windows 11 or previous versions of the software. The license is unique to that specific piece of software.

Are EULAs and Software Licenses different? It depends on who you ask, as they can cover many of the same topics. For what topics we are covering, it is pretty safe to equate an End User License Agreement to various other Software Licenses. One main difference is EULAs tend to not differentiate between source code and compiled code, where open-source licenses may be explicit on those two topics.

Source Sharing

While many proprietary licenses restrict gaining access to the source code of a particular piece of software, that is not a defining factor of a proprietary license. Some software may come with an option called Source Sharing, where a software provider allows a customer access to the source code.

I have come across this mostly in enterprise software where the software provider sells software that solves many common problems for their customers, but customers may have very unique workflows or requirements. I used to work in the insurance industry, and all of our back-office software came with a source-sharing agreement. Each release of the software included a full copy of the source code we would patch with our custom changes and compile on our systems.

We would modify the software to work exactly how we needed it and work with the vendor and other customers to get our patches merged into the mainline code by the vendor. Customers were allowed to share their patches with the system among themselves. It was very much like a limited open-source ecosystem.

Where source sharing differs from Open Source licensing, the original vendor still controls the source. While we were granted access to the software, we could only share it with other customers. We were still required to pay a hefty licensing fee to get access to the software. Most damning of all, we found out after doing a license audit that the changes we made had an automatic copyright transfer to the vendor.

What that meant was that the original vendor could, at any time, take our patches and sell them as their own. They could even take our patches and lock them behind an even more expensive license. While we had access to the source code, the proprietary license dictated that we did not own any of the code, even the code we wrote.

Open Source Licensing

Many people do not realize that despite open-source software arguably being the original way software was distributed, the term "Open Source Software" was not codified until 1998. This official definition is called "The Open Source Definition[8]" and was published by the Open Source Initiative as a copy of the Debian Free Software Guidelines[9]. These rules specify what makes a piece of software Open Source.

A collective known as the Open Source Initiative[10] is a group that helps govern and guide open-source software. This includes maintaining the "Open Source Definition" as well as providing various resources for open source projects. One of their most important projects is a list of open-source licenses that they consider compatible with the idea of Open Source.

For a piece of software to be considered open source, it must meet the following guidelines[11] from Wikipedia:

1. **Free redistribution**: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Source code**: The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
3. **Derived works**: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of the author's source code**: The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No discrimination against persons or groups**: The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavor**: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. **Distribution of license**: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License must not be specific to a product**: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
9. **License must not restrict other software**: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
10. **License must be technology-neutral**: No provision of the license may be predicated on any individual technology or style of interface.

Smelly Open Source

Much like Public Domain-like licensing, many licenses look like open-source licenses but actually restrict what the user can do. The JSON License[12] is a perfect example with its famous line, "The Software shall be used for Good, not Evil."

What is legally Good, and what is legally Evil? Even outside of the legal definitions, an open-source license should not, and cannot, restrict the user in such arbitrary ways. A truly open source software license does not restrict what the user can do nor force the user to do specific things. Rule #6 of the Open Source Definition expressly invalidates anything licensed under licenses like the JSON license.

8 The Open Source Definition: https://en.wikipedia.org/wiki/The_Open_Source_Definition

9 Debian Free Software Guidelines: https://w.wiki/4z9Q

10 Open Source Initiative: https://opensource.org

11 guidelines: https://w.wiki/4z9P

12 The JSON License: https://www.json.org/license.html

Permissive Licenses

Permissive licenses are licenses that are not copyleft licenses (more about them in a moment) and allow proprietary derivative works. In many ways, this mirrors how software was originally handled—a developer puts software out into the world and allows anyone else to use it, even if that code gets locked up in proprietary software. The general idea is that the software being available makes the developer's life easier.

A few of the most common permissive licenses are the MIT[13], BSD[14], and Apache 2.0 licenses[15]. The MIT and BSD licenses closely resemble each other, though there are a variety of BSD licenses that have come out through the years. This general format is what has inspired many of the "smelly" open source licenses and shorter licenses.

You may see the BSD license also called the Original BSD License, and anywhere from the 0 Clause BSD License to a 4 Clause BSD license. Over the years, various rules surrounding licensed software have changed. For example, the 2 Clause BSD license drops a non-endorsement requirement that the 3 Clause BSD license includes. The 3 Clause BSD dropped an advertising requirement that the 4 Clause BSD license imposed. These days the 2 or 3 Clause BSD license is typically used.

Personally, I see permissive licenses working best in library or component code or places where the code is clearly intended to be used by other code. For example, I release my dragonmantank/cron-expression[16] library under the MIT license because it is meant to be used with someone else's code. I am more interested in solving a problem for a developer rather than making sure that the developer releases any changes back into the wild.

The fact that permissive licenses do allow for proprietary usage is a major downside to this type of license. One of the most famous examples of this was that Windows 2000 contained BSD licensed code[17] as part of the networking tools and stack. People were shocked at this, but Microsoft was well within their rights to use the code as long as they followed the license. And they did. If you are OK with allowing your code to be used this way, permissive licenses are a good selection for your code.

Copyleft Licenses

This brings us to Copyleft licenses. Copyleft licenses differ from permissive licenses in that they require derivative software to be licensed under the license of the software that was being integrated. As mentioned above, it is perfectly acceptable for permissive-licensed code to be used in software that does not share that same license (ala BSD code being used in proprietary code). On the other hand, copyleft software makes it a requirement that the code stay under the same license. Copyleft licenses tend to cater toward software freedom more than developer freedom.

The GPL[18] is usually the go-to example of a copyleft license. The GPL itself was born out of the frustration that proprietary software was causing to developers and how software was increasingly stripping developers of the rights they used to have. As part of this, the requirement that GPL-derived software must also be GPL licensed was a conscious decision. This decision forced developers that altered the software to distribute those changes when someone asked. In fact, a derivative license called the Affero GPL[19] (AGPL) even goes so far as to say that anyone that simply accesses the code can ask for the source code changes.

I find that copyleft licenses, especially the GPL itself, work best for full applications. Since applications tend to solve much larger problems and generally involve a huge amount of developer hours, it makes much more sense to keep ill actors from taking the open-source software and just renaming it and slapping a proprietary label. Imagine if the Linux kernel was released under the MIT license. Well, we know what happens.

Two popular operating systems are based on BSD code:

1. The macOS base operating system called Darwin[20]
2. The Orbis OS[21] that powers the Playstation 4 and Playstation 5

While Darwin started out very open, Apple increasingly slowed down upstream patches to the OS. While Apple is very upfront about its use of BSD software, very few realize that the Playstation is running a BSD-powered operating system. Sure, both Sony and Apple are legally following what the BSD license tells them to, but there is a vast amount of work that neither company is required to release back to the ecosystem.

So What do You do?

When you go to release software, think about your goal for users. Ask yourself this question, "should the software be proprietary or open-source?" While I wholeheartedly support open source and believe it is the way software should be distributed, I use an iPad and iPhone, and I use Windows for a lot of video gaming. I do a lot of open source work, but the vast majority of my life has been spent making proprietary software.

If you release software as open-source, do it the proper way. Use a license approved by the Open Source Initiative, and make sure you follow the Open Source Definition for your

13 *MIT: https://opensource.org/licenses/MIT*

14 *BSD: https://opensource.org/licenses/BSD-3-Clause*

15 *Apache 2.0 licenses: https://opensource.org/licenses/Apache-2.0*

16 *dragonmantank/cron-expression: https://github.com/dragonmantank/cron-expression*

17 *BSD licensed code: https://phpa.me/everything2-bsd-windows*

18 *GPL: https://opensource.org/licenses/GPL-3.0*

19 *Affero GPL: https://opensource.org/licenses/AGPL-3.0*

20 *Darwin: https://w.wiki/4wKH*

21 *Orbis OS: https://www.extremetech.com/?p=159476*

software. It is not much work, and you will find a license that fits your software's goals.

If you are just a developer, keep in mind what software you use and make sure you follow the license. Pay particular attention to what the dependencies of your dependencies require. Comcast has a license checker[22] that you can use to scan your `composer.lock` file to help suss out this information. Understand what it means to consume software under the different licenses.

I hope all this information helps clear up a lot of the misconceptions and unknown pitfalls of licensing. I would love to live in a world where we just share code and do not have to worry about the legalities of software design, but this is the world we live in. All I can say is help spread open-source software, and use it responsibly.

22 license checker: https://github.com/Comcast/php-legal-licenses

Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. Chris has worked with many different frameworks and languages throughout his twelve years of programming 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

Related Reading

- *PHP is the Worst* by Chris Tankersley, September 2021
  https://phpa.me/tankersley-sept-2021
- *How to Build a REST API* by Chris Tankersley, May 2021
  https://phpa.me/tankersley-may-2021
- *Overriding Composer* by Chris Tankersley, October 2019
  http://phpa.me/education-oct-19

Honeybadger provides all the context you need to understand what is causing an exception

Honeybadger.io

The Web Developer's

SECRET WEAPON!

Exception, uptime, and cron monitoring, all in one place and easily installed in your web app. Deploy with confidence and be your team's devops hero.

Are exceptions *all* that keep you up at night?

Honeybadger gives you full confidence in the health of your production systems.

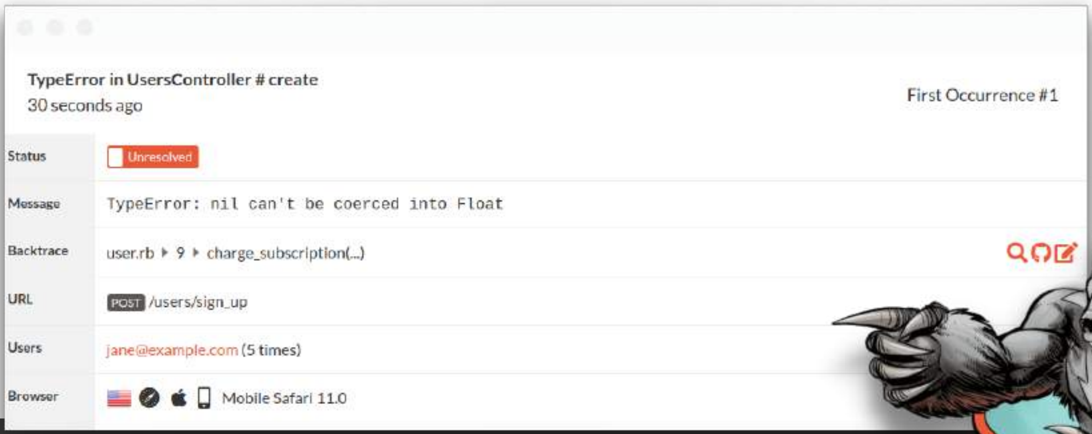
DevOps monitoring, for developers. *gasp!*

Deploying web applications at scale is easier than it has ever been, but monitoring them is hard, and it's easy to lose sight of your users. Honeybadger simplifies your production stack by combining three of the most common types of monitoring into a single, easy to use platform.

Exception Monitoring
Delight your users by proactively monitoring for and fixing errors.

Uptime Monitoring
Know when your external services go down or have other problems.

Check-In Monitoring
Know when your background jobs and services go missing or silently fail.



TypeError in UsersController # create
30 seconds ago

First Occurrence #1

Status	Unresolved
Message	`TypeError: nil can't be coerced into Float`
Backtrace	user.rb ▸ 9 ▸ charge_subscription(...)
URL	POST /users/sign_up
Users	jane@example.com (5 times)
Browser	Mobile Safari 11.0

Start Your Free Trial Today

https://www.honeybadger.io/