Debugging with Xdebug and Sublime Text 3

By Peter Nijssen

PHP

February 26, 2014

Share:

Debugging – we all do it a lot. Writing code perfectly the first time around is hard and only a few (if any) succeed at it. More than a year ago, Shameer wrote an article on SitePoint about how you can debug your application using Xdebug and Netbeans. In this article, we are going to have a look at how we can debug using Xdebug in combination with Sublime Text.

Getting started

First of all, we need to have the PHP Xdebug extension installed. If you are uncertain on how to get this done, please have a look at the link provided in the introduction. Make sure that Xdebug is working by checking if it's listed in your `phpinfo()`.
Of course we also need Sublime Text. I will be using the latest version: Sublime Text 3. It should also work with Sublime Text 2.

Setting up Xdebug

We need to configure xdebug by adding the following to your `php.ini` file, or even better, to an `xdebug.ini` file as described here under How-to On Linux.

```
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
```

```
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
xdebug.remote_log="/var/log/xdebug/xdebug.log"
```

In general you will be using 127.0.0.1 as your host. However, If you are using vagrant for example, you will be using something like 10.0.2.2, depending on where Xdebug can find your system.

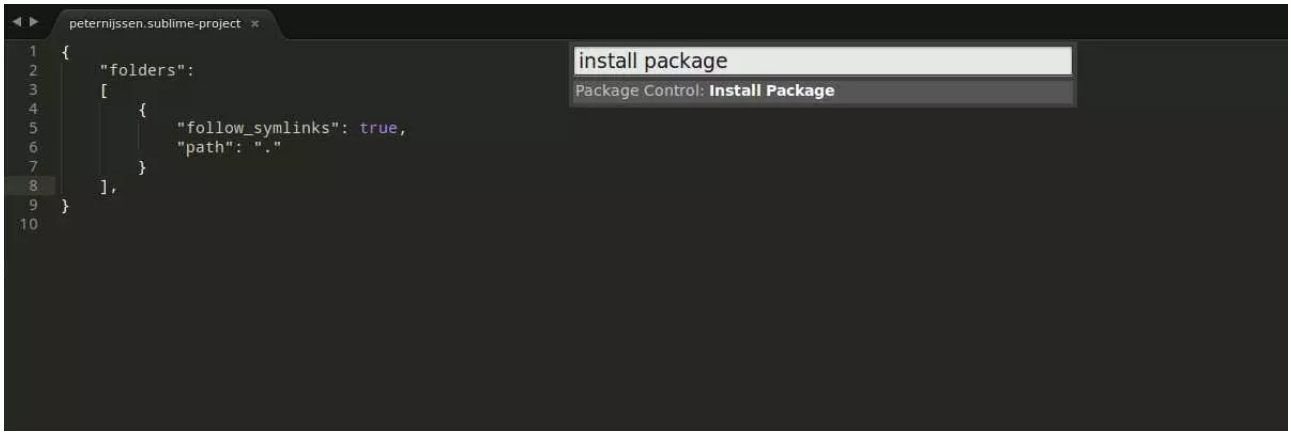
The remote log is not necessary, but in case of problems, it's the place where you can find information about errors that occurred.

Don't forget to restart your webserver!

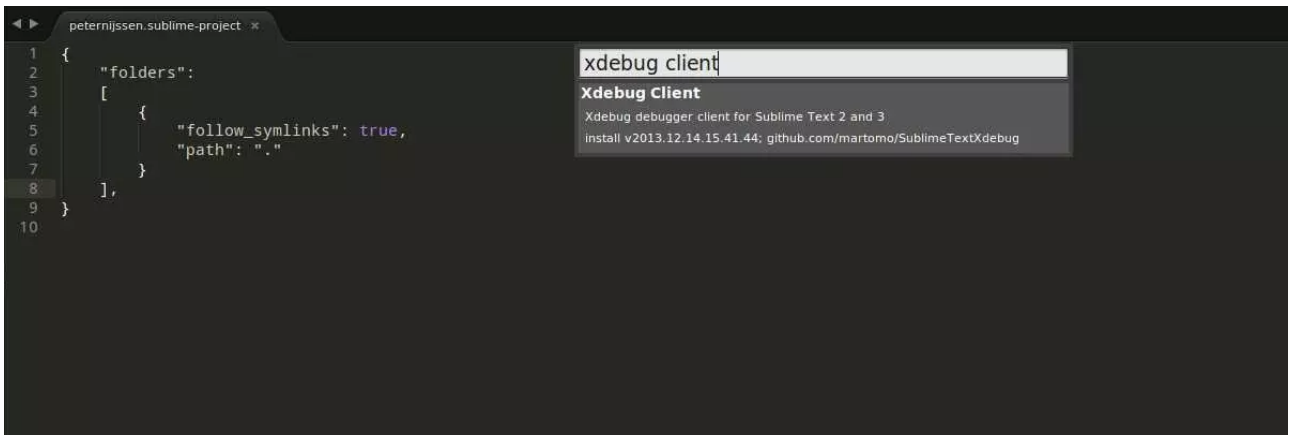
Setting up Sublime Text 3

One of the strengths of Sublime is the fact that you can extend it easily with packages. In this case, we are going to install the Xdebug package. If you haven't done so already, make sure you can install packages by installing package control.

Once you have the package control installed, you should start Sublime Text 3. Open up the command palette from the tools menu and search for "install package".



Now you can search for any package you like. In our case, we are going to search for the package “Xdebug client”.



The last bit we have to do is set up the project within Sublime. The easiest way to do this is to open up the root directory of your application, go to projects and click on “save projects as”. I suggest you save the file within the root of your application, so you can save it in your version control system if you are using any and you can configure it easily at all times.

Open up the just created project file. The content will look like this:

```
{
    "folders":
    [
        {
            "follow_symlinks": true,
            "path": "."
        }
    ]
}
```

We are going to add a few more lines:

```
{
    "folders":
    [
        {
            "follow_symlinks": true,
            "path": "."
        }
    ],
    "settings": {
        "xdebug": {
             "url": "http://my.local.website/",
        }
    }
}
```

As you can see, I only added a URL to my actual web application. I could set more settings for Xdebug, however, this is enough to start with. I could have also set this URL in the Xdebug settings itself, but in that case, I couldn't work on multiple projects without having to change the Xdebug config each time.

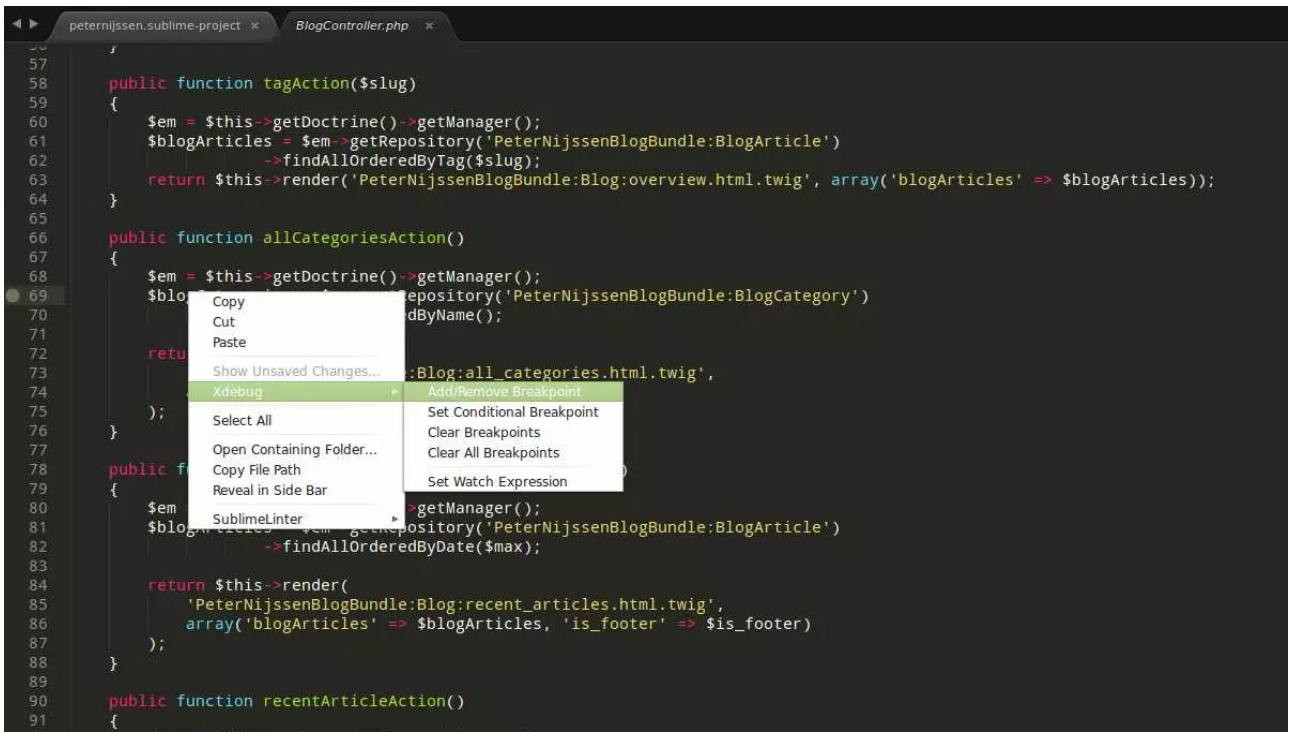
Start the Xdebug session

We can now start the Xdebug session to see if everything is set up properly. In the menu, click on tools -> Xdebug and click on start debugging (launch browser). You will notice that your website is opened up and that `?XDEBUG_SESSION_START=sublime.xdebug` is added to the end of the URL. This will start the xdebug session. In Sublime, some extra panels appear where debug information will be shown, after you have set one or more breakpoints.

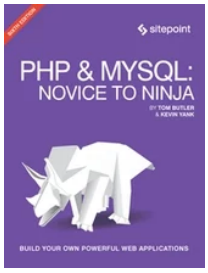
Breakpoints

Let's set out first breakpoint. A breakpoint is basically a flag where your application will halt when it reaches it. At the moment it halts, you can inspect all the variables' values so you know actually what is going on.

We can add a breakpoint by clicking with our right mouse on a line, going to Xdebug and then clicking on add/remove breakpoint. A marker will be added to the line gutter to indicate that a breakpoint has been set.

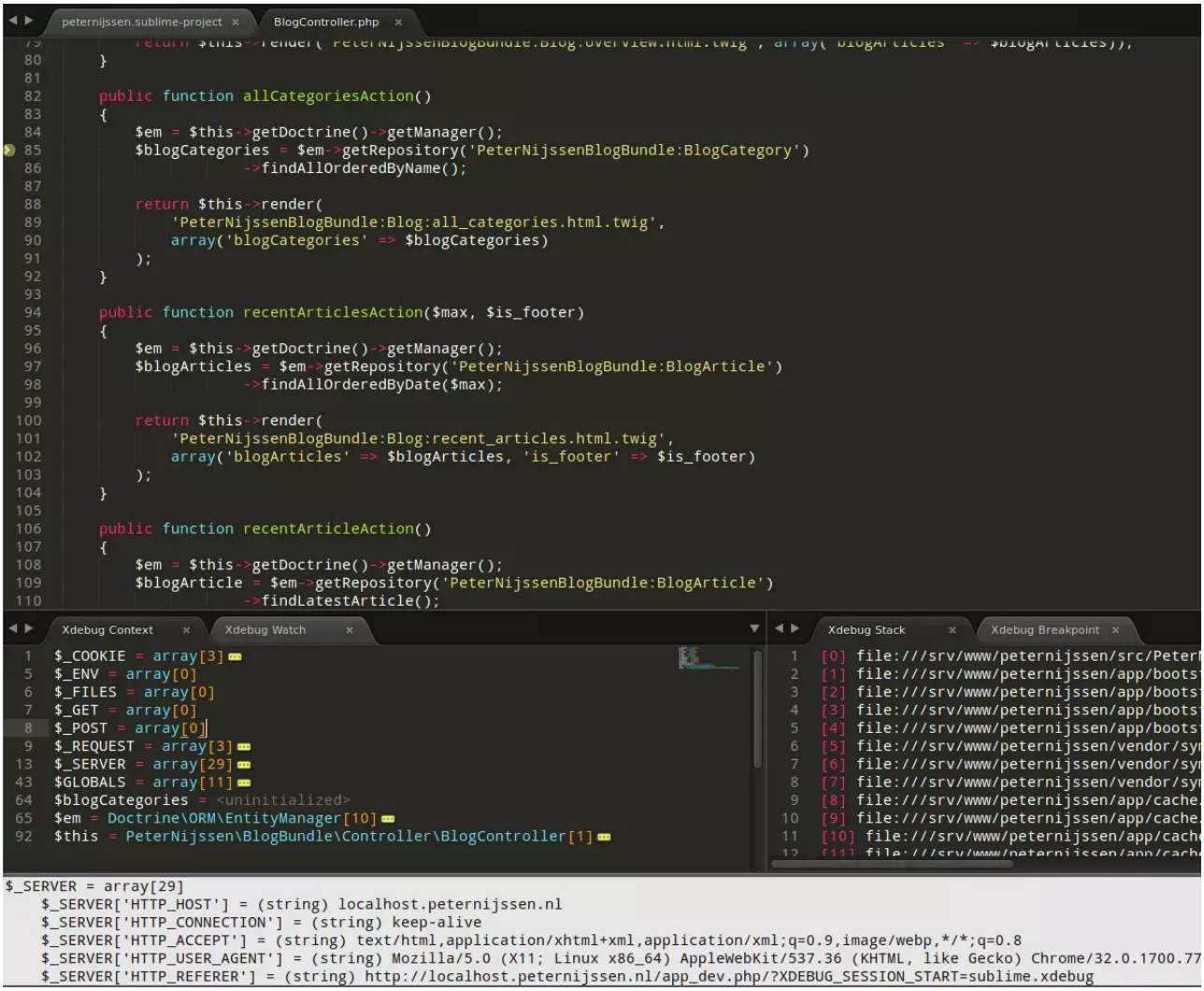


We open up our browser again and continue with the session we just started. You will notice that as soon as you go to the page where the breakpoint is, the page will stop loading. If you now open up Sublime, you will see a lot of information shown in the Xdebug panels.



The Xdebug stack and Xdebug context are very interesting. In the stack, you can see the whole stacktrace your call went through.

In the context, you will see all global variables, but also the variables you defined yourself. You can click on these variables to see exactly these variables are holding. For instance, in the screenshot below, I clicked on the $_SERVER variable.



Notice that a yellow arrow is pointing at the line the application is currently halted on.

So our application halted and now we can look through the variables defined. However, we are done and we want to move on. What now? When you right mouse click once again and hover over the Xdebug menu, you will have several options:

- **Run** Which will run the application until the next breakpoint or until the ending.
- **Run to line** which will run until the line you clicked.
- **Step into** will step into the current function and stops right after.
- **Step over** Will step over the current function and stops right after.
- **Step out** Will step out of the current function and stop right after.

- **Stop** Will stop debugging.
- **Detach** Will also stop debugging.

Run and stop are quite easy to understand. The step methods could be a little confusing. Let's dive into these with a simple example.

```
Class Foo()
{

    public function bar(Array $arr)
    {
        $arr = self::fooBar($arr); // Breakpoint
        return $arr;
    }

    public function fooBar(Array $arr)
    {
        return array_values($arr);
    }
}
```

Imagine you added a breakpoint to the first line of the method bar. So on the line with the breakpoint comment (`// breakpoint`).

With step into, the debugger will step into the fooBar method and will stop there at the first line. So in this case, the debugger will halt on the `return array_values($arr);` line.

Peter Nijssen

Peter is a software architect from the Netherlands. He freelanced for more then 6 years as a web developer, and meanwhile, he graduated as software engineer with honors. He decided to join CMNTY Corporation

which specializes in creating community software and is now responsible for the ongoing development of multiple web applications as well as mobile applications. Peter believes a real developer is able to combine multiple techniques together to make sure the user receives the ultimate experience and enjoys using the application. In his free time, he loves to play board games with anyone who is interested. He especially has a passion for cooperative board games.

Popular Books

Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers

Your First Year in Code

Jump Start Git, 2nd Edition

How to Ensure Flexible, Reusable PHP Code with Insphpect

By Tom Butler

PHP

June 30, 2020

Share:

Insphpect **is a tool I wrote as part of my PhD project. It scans code for object-oriented programming techniques that hinder code reusability and flexibility.**

Why?

Let me begin with two mundane observations:

1. Business requirements change over time.
2. Programmers are not clairvoyant.

New product launches, emergency lockdown regulations, expanding into new markets, economic factors, updated data protection laws: there are lots of potential causes for business software to need updating.

From those two observations we can infer that programmers know that the code they write is going to change, but not what those changes will be or when they will happen.

Writing code in such a way that it can be easily adapted is a skill that takes years to master.

You're probably already familiar with programming practices that come back and haunt you. Novice programmers quickly realize that global variables are more trouble than they're worth, and the once incredibly popular Singleton Pattern has been a dirty word for the last decade.

How you code your application has a big impact on how easy it is to adapt to meet new requirements. As you progress through your career, you learn techniques that make adapting code easier. Once you've grasped fundamentals of object-oriented programming you wonder how you ever did without it!

If you ask ten developers to produce software, given the same requirements, you'll get ten different solutions. Some of those solutions will inevitably be better than others.

Consider a ship in a bottle and a model ship made of Lego. Both are model ships, but changing the sails on the ship in a bottle is very difficult, and reusing the parts is near impossible. However, with a Lego ship, you can easily swap out the sails or use the same components to build a model rocket, house or a car.

Certain programming techniques lead to the *ship-in-a-bottle* approach and make your code difficult to change and adapt.

Insphpect

Insphpect is a tool which scans your code for programming practices that lead to this kind of a ship in a bottle design.

It grades your code based on how flexible it is, and highlights areas where flexibility can be improved.

What does Insphpect look for?

Currently, Insphpect looks for the following:

- tight coupling
- hardcoded configuration
- singletons
- setter injection
- using the `new` keyword in a constructor
- service locators
- inheritance
- static methods
- global state
- files that have more than one role (e.g. defining a class and running some code)

If it detects anything it identifies as inflexible, it highlights the code, explains why it highlighted the issue, then grades your whole project and individual classes on a score of 0-100 (with 100 being no issues detected). As a proof of concept, for some detections it's able to automatically generate a patch file that re-writes the code to remove the inflexibility entirely.

Take a look a sample report here.

Insphpect is currently in the testing phase, and it would really help my research progress if you can check it out and complete the survey in the “Give your feedback” section of the site.

Background

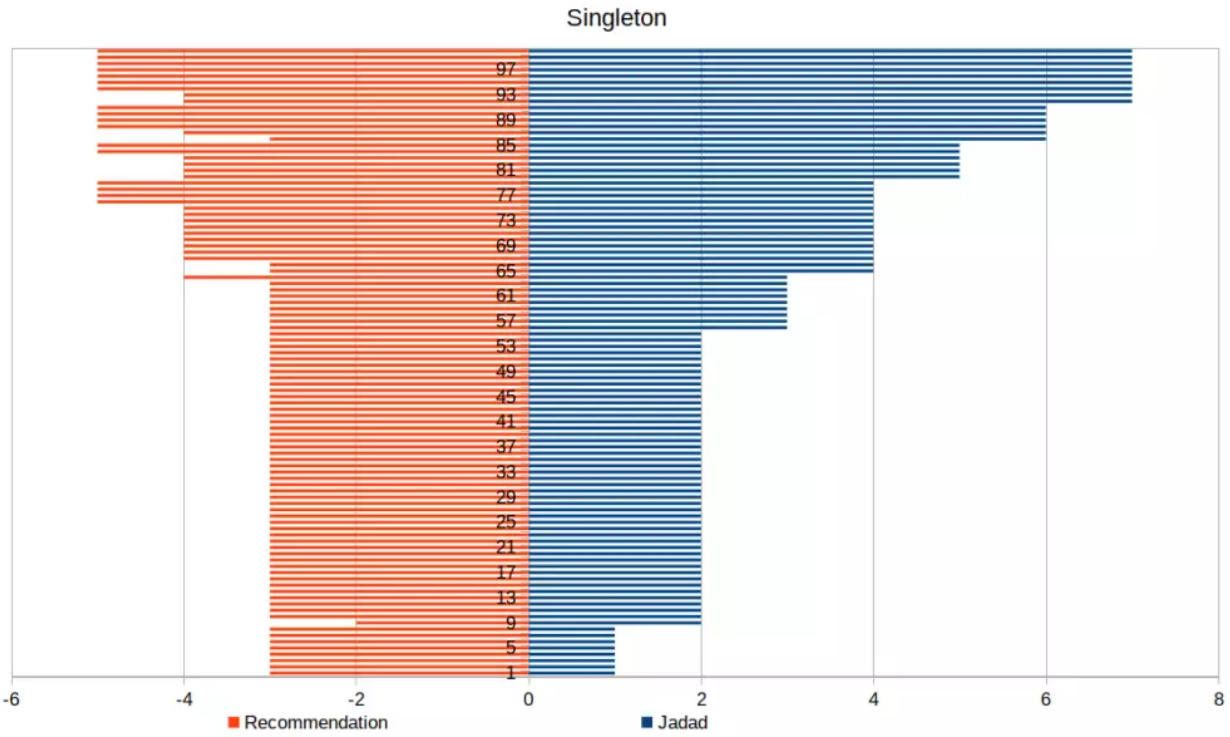
Are those bad practices really bad, though?

This was one of the more difficult parts of the background research, and you can read about how this was done in detail on the Insphpect website.

However, this can be summarized as:

- The opinions of each bad practice were collected from 100 authors per practice.
- The author’s opinion on the practice was graded on a scale of 1–5.
- The author’s methodological rigor was graded on a scale of 1–7 based on the Jadad score used for clinical trials.

These were then plotted like the graph below:



Each horizontal line represents an article, and the left (orange) bar for each article is the recommendation going from 5 — Avoid this practice at all costs (Far left) — to 1 — Favor this practice over alternatives.

The right (blue) bar for each article is the Jadad style score measuring analytic rigor. A score of seven means the article describes the practice, provides code examples, discusses alternative approaches, provides like-for-like code samples, discusses the pros/cons of each approach and makes a recommendation of which approach should be used.

In the case of the singleton above, authors who compare the singleton to alternative approaches, discuss the pros/cons, etc., are significantly more likely to suggest using alternative approaches.

Walkthrough

Currently, Insphpect allows uploading code via a Git repository URL or a ZIP file.

So not to point out flaws in other people's work, let's take a look at one of my own projects to see what it identifies.

We'll use https://github.com/Level-2/Transphporm as an example project.

This is quite a good example, because it has a very high score on another code-quality tool Scrutinizer.

Firstly, enter the git URL `https://github.com/Level-2/Transphporm` into the text box at the top of the home page and press "Go". It will take a few seconds to minutes, depending on the size of the project, and will generate a report that looks something like this:

Once you're on the report page, you'll see a summary at the top with an overall grade out of 100, with 100 being very good and 0 being very poor.

Underneath the summary, you'll see a list of all the classes in the project, each with its own grade.

Don't worry if your code doesn't get a perfect score. It's unlikely that it will. Remember, Insphpect is a tool that identifies flexibility in your code. There are parts of your code (like the entry point) where flexibility isn't warranted.

For Transphporm, it has highlighted issues in seven classes.

Let's take a look at some of those. Scroll down to `Transphporm\Parser\CssToXpath` and click the link. You'll see a score for that particular class and a list of issues which have been identified.

In this case, it has identified a static variable and a static method. Clicking on one of the red lines will reveal an explanation of why the line was flagged up.

For example, clicking line 12 will give an explanation of why static variables are less flexible than instance variables.

Although there's a more in-depth explanation of the issues caused by static properties on the report, as a quick refresher, static variables have one value which is shared across all the instances of the class.

This is inherently less flexible than an instance variable, because using an instance variable allows each instance to have a different value.

For example, consider the following:

```
class User {
    public static $db;
    public $id;
    public $name;
    public $email;
```

```
    public function save() {
        $stmt = self::$db->prepare('REPLACE INTO user (id, name, email) \

        $stmt->execute([
            'id' => $this->id,
            'name' => $this->name.
            'email' => $this->email
        ]);
    }
}
```

Because `$db` is static, every instance of this class shares the same `$db` instance and records will always be inserted into the same database.

While this sounds reasonable, let me give you a real-world example.

In the real world

One of our clients was a recruitment agency. About two years after we developed their site, they took over another smaller company. They wanted to retain the second company's website and branding because it was quite well known in the niche they were in.

Our client asked us the following:

> On the second company's site, can you add a checkbox when adding a job that also adds the job to our database so people viewing our site can also see the job and visa versa.

A fairly simple request. Run an insert query into two different databases.

But because the website used a static global database instance, this was needlessly difficult!

The developers of that site wrote the code confident that only one database connection would ever be needed. They were wrong.

Remember, you're not clairvoyant, and it's impossible to anticipate what flexibility may be needed in the future.

The solution

As suggested by Insphpect, the solution to this is using instance variables:

```
class User {
    private $db;
    public $id;
    public $name;
    public $email;

    public function __construct(\PDO $db) {
        $this->db = $db;
    }

    public function save() {
        $stmt = self::$db->prepare('REPLACE INTO user (id, name, email) \

        $stmt->execute([
            'id' => $this->id,
            'name' => $this->name.
            'email' => $this->email
        ]);
    }
}
```

Now a `User` instance can be used with different database instances:

```
new User($database1);
new User($database2);
```

For `Transphporm\Parser\CssToXpath` we could do the same, remove the static variable and consider making it an instance variable rather than a static variable.

Using new in constructor

Let's take a look at one of the other classes: `Transphporm\Builder`.

This has a score of zero, which is rather poor. Examining the report in detail, Insphpect has picked up the same issue three times: using the `new` keyword in a

constructor.

Google Programming Coach Misko Hevery does a great job at explaining why this is a poor programming practice, but here's a simple example from Insphpect's output:

```
class Car {
    private $engine;

    public function __construct() {
        $this->engine = new PetrolEngine();
    }
}
```

Here, whenever an instance of `Car` is created, an instance of `PetrolEngine` is created. That makes it very inflexible, because there 's no way to construct a `Car` with a different engine type. Every car modeled in this system must have a `PetrolEngine`.

Instead, we can use dependency injection:

```
class Car {
    private $engine;

    public function __construct($engine) {
        $this->engine = $engine;
    }
}
```

Different cars can be created with an instance of `PetrolEngine`, `DieselEngine`, `ElectricEngine`, `JetEngine` or any other engine type that exists in the project.

To fix this error in the `Transphporm\Builder`, all of the variables that currently have hard-coded class names should use constructor arguments instead.

There are other issues identified by Insphpect, but you can try it out for yourself and see how your project fares.

Behind the Scenes

You might be wondering how the scores are calculated and why this class got a zero. At the present time, the weightings are subject to change once more projects have been scanned and more feedback has been provided.

The scores are designed to be indicative for comparing one project/class to another.

The overall project score is just an average of all the classes in the project. This was implemented because a project with two issues in 1000 classes is a lot better overall than a project with two issues in two classes.

Each bad practice is weighted based on whether it impedes flexibility for the entire class or only impedes flexibility for a method.

Conclusion

Insphpect can be used to identify areas of your code which make future changes more difficult than they could be, and it offers suggestions on how to write the code in a more flexible manner. Remember, you're not clairvoyant and have no way to know how your code is going to need to change!

Insphpect is currently a work in progress, and the more people who use it (and complete the survey) the better it will become.

How did your project or favorite library score? Be sure to complete the survey, as it will provide valuable data for my PhD project and help the tool improve!

Tom Butler

Tom Butler is a Web Developer, Ph.D student researching software best practices, and part-time University Lecturer from the UK with an interest in programming best practices, separation of concerns and a 'less is more' approach to code.

Popular Books

Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers

Your First Year in Code

Jump Start Git, 2nd Edition

- Remote Jobs
- Advertise

Connect