

ENTROPY WINS ([HTTPS://WWW.ENTROPYWINS.WTF/BLOG/](https://www.entropywins.wtf/blog/))

A blog on Software Architecture, Design and Craftsmanship

Clean Code

(<https://www.entropywins.wtf/blog/tag/clean-code/>)

Craftsmanship (<https://www.entropywins.wtf/blog/tag/software-craftsmanship/>)

Architecture (<https://www.entropywins.wtf/blog/tag/software-architecture/>)

Testing (<https://www.entropywins.wtf/blog/tag/testing/>)

PHP (<https://www.entropywins.wtf/blog/tag/php/>)

Open Source (<https://www.entropywins.wtf/blog/tag/open-source/>)

MediaWiki (<https://www.entropywins.wtf/blog/tag/mediawiki/>)

Jeroen De Dauw (<https://www.entropywins.wtf/>)

MediaWiki Hosting (<https://www.pro.wiki/>)

Professional.Wiki (<https://professional.wiki/>)

 FOLLOW ME ON TWITTER ([HTTPS://TWITTER.COM/JEROENDED AUW](https://twitter.com/jeroendedauw))

 FOLLOW ME ON GITHUB ([HTTPS://GITHUB.COM/JEROENDED AUW](https://github.com/jeroendedauw))

FOLLOW ME ON DEV ([HTTPS://DEV.TO/JEROENDED AUW](https://dev.to/jeroendedauw))

RSS ([HTTPS://WWW.ENTROPYWINS.WTF/BLOG/FEED/](https://www.entropywins.wtf/blog/feed/))

ATOM ([HTTPS://WWW.ENTROPYWINS.WTF/BLOG/FEED/ATOM/](https://www.entropywins.wtf/blog/feed/atom/))

PRIVACY POLICY ([HTTPS://WWW.ENTROPYWINS.WTF/PRIVACY-POLICY](https://www.entropywins.wtf/privacy-policy))

The Fallacy of DRY

🕒 2017-09-06 👤 Jeroen (<https://www.entropywins.wtf/blog/author/jeroen/>) 💬 13
Comments (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comments>)

DRY, standing for Don't Repeat Yourself

(https://en.wikipedia.org/wiki/Don%27t_repeat_yourself), is a well-known design principle in the software development world.

It is not uncommon for removal of duplication to take center stage via mantras such as “Repetition is the root of all evil”. Yet while duplication is often bad, the well intended pursuit of DRY tends to lead people astray. To see why, let's take a step back and look at what we want to achieve by removing duplication.

THE GOAL OF SOFTWARE

First and foremost, software exists to fulfill a purpose. Your client, which can be your employer, is paying money because they want the software to provide value. As a developer it is your job to provide this value as effectively as possible. This includes tasks beyond writing code to do whatever your client specifies, and might best be done by not writing any code. The creation of code is expensive. Maintenance of code and extension of legacy code is even more so.

Since creation and maintenance of software is expensive, the quality of a developers work (when just looking at the code) can be measured in how quickly functionality is delivered in a satisfactory manner, and how easy to maintain and extend the system is afterwards. Many design discussions arise about trade-offs between those two measures. The DRY principle mainly situates itself in the latter category: reducing maintenance costs. Unfortunately applying DRY blindly often leads to increased maintenance costs.

THE GOOD SIDE OF DRY

So how does DRY help us reduce maintenance costs? If code is duplicated, and it needs to be changed, you will need to find all places where it is duplicated and apply the change. This is (obviously) more difficult than modifying one place, and more error prone. You can forget about one place where the change needs to be applied, you can accidentally apply it differently in one location, or you can modify code that happens to be the same at present but should nevertheless not be changed due to conceptual differences (more on this later). This is also known as Shotgun Surgery (https://en.wikipedia.org/wiki/Shotgun_surgery).

Duplicated code tends to also obscure the structure and intent of your code, making it harder to understand and modify. And finally, it conveys a sense of carelessness and lack of responsibility, which begets more carelessness.

Everyone that has been in the industry for a little while has come across horrid procedural code, or perhaps pretend-OO code, where copy-paste was apparently the favorite hammer of its creators. Such programmers indeed should heed DRY, cause what they are producing suffers from the issues we just went over. So where is The Fallacy of DRY?

THE FALLACY OF DRY

Since removal of duplication is a means towards more maintainable code, we should only remove duplication if that removal makes the code more maintainable.

If you are reading this, presumably you are not a copy-and-paste programmer. Almost no one I ever worked with is. Once you know how to create well designed OO applications (ie by knowing the SOLID principles ([https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)))), are writing tests, etc, the code you create will be very different from the work of a copy-paste-programmer. Even when adhering to the SOLID principles (to the extend that it makes sense) there might still be duplication that should be removed. The catch here is that this duplication will be mixed together with duplication that should stay, since removing it makes the code less maintainable. Hence trying to remove all duplication is likely to be counter productive.

Costs of Unification

How can removing duplication make code less maintainable? If the costs of unification outweigh the costs of duplication, then we should stick with duplication. We've already gone over some of the costs of duplication, such as the need for Shotgun Surgery. So let's now have a look at the costs of unification.

The first cost is added **complexity**. If you have two classes with a little bit of common code, you can extract this common code into a service, or if you are a masochist extract it into a base class. In both cases you got rid of the duplication by introducing a new class. While doing this you might reduce the total complexity by not having the duplication, and such extracting might make sense in the first place for instance to avoid a Single Responsibility Principle violation. Still, if the only reason for the extraction is reducing duplication, ask yourself if you are reducing the overall complexity or adding to it.

Another cost is **coupling**. If you have two classes with some common code, they can be fully independent. If you extract the common code into a service, both classes will now depend upon this service. This means that if you make a change to the service, you will need to pay attention to both classes using the service, and make sure they do not break. This is especially a problem if the service ends up being extended to do more things, though that is more of a SOLID issue. I'll skip going over the results of code reuse via inheritance to avoid suicidal (or homicidal) thoughts in myself and my readers.

DRY = Coupling

— A slide at DDDEU 2017

The coupling increases the need for **communication**. This is especially true in the large, when talking about unifying code between components or application, and when different teams end up depending on the same shared code. In such a situation it becomes very important that it is clear to everyone what exactly is expected from a piece of code, and making changes is often slow and costly due to the communication needed to make sure they work for everyone.

Another result of unification is that code can no longer **evolve separately**. If we have our two classes with some common code, and in the first a small behavior change is needed in this code, this change is easy to make. If you are dealing with a common service, you might do something such as adding a flag. That *might* even be the best thing to do, though it is likely to be harmful design wise. Either way, you start down the path of **corrupting your service**, which now turned into a frog in a pot of water that is being heated. If you unified your code, this is another point at which to ask yourself if that is still the best trade-off, or if some duplication might be easier to maintain.

You might be able to represent two **different concepts** with the same bit of code. This is problematic not only because different concepts need to be able to evolve individually, it's also **misleading** to have only a single representation in the code, which effectively hides that you are dealing with two different concepts. This is another point that gains importance the bigger the scope of reuse. Domain Driven Design has a strategic pattern called Bounded Contexts (<http://martinfowler.com/bliki/BoundedContext.html>), which is about the separation of code that represents different (sub)domains. Generally speaking it is good to avoid sharing code between Bounded Contexts. You can find a concrete example of using the same code for two different concepts in my blog post on Implementing the Clean Architecture (<https://www.entropywins.wtf/blog/2016/11/24/implementing-the-clean-architecture/>), in the section “Lesson learned: bounded contexts”.

DRY is for one Bounded Context

— Eric Evans in *Good Design is Imperfect Design*
(<https://www.youtube.com/watch?v=ly54TmmElly>)

EXAMPLE

```
1 for($i=0; $i<4, ++$i) {  
2     doAction($i);  
3 }
```

How many times is `doAction` called? 3 times? 4 times? What about in this snippet:

```
1 doAction(1);  
2 doAction(2);  
3 doAction(3);
```

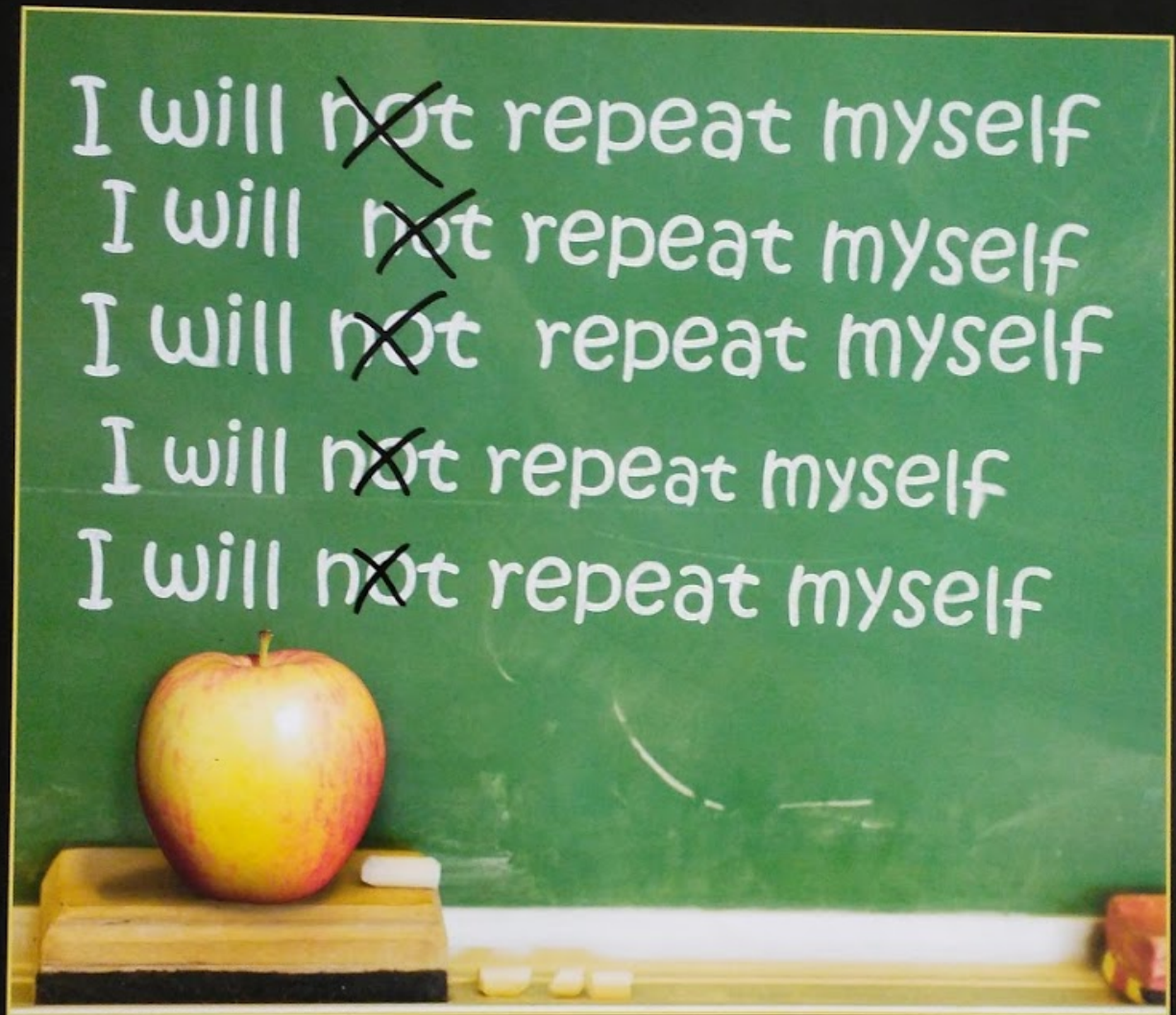
I'm sure you can figure out what the first snippet does. The point here is that it takes more effort and that it is a lot easier to make a mistake.

This demonstrates how removing duplication on a very low-level can be harmful. If you have a good higher level example that can be understood unambiguously without providing a lot of extra context, do leave a comment.

CONCLUSION

Duplication itself does not matter. We care about code being easy (cheap) to modify without introducing regressions. Therefore we want simple code that is easy to understand (<https://www.entropywins.wtf/blog/2017/01/02/simple-is-not-easy/>). Pursuing removal of duplication as an end-goal rather than looking at the costs and benefits tends to result in a more complex codebase, with higher coupling, higher communication needs, inferior design and misleading code.

You can read a different style of explanation in the blog post by Sandi Metz on the same topic titled The Wrong Abstraction (<https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction>).



DO REPEAT YOURSELF

AVOIDING Repetition is A root of software COMPLEXITY

Post Views: 46,907

← [Review of Ayreon: The Source](https://www.entropywins.wtf/blog/2017/06/06/review-of-ayreon-the-source/)

[\(https://www.entropywins.wtf/blog/2017/06/06/review-of-ayreon-the-source/\)](https://www.entropywins.wtf/blog/2017/06/06/review-of-ayreon-the-source/)

[Yield in PHPUnit data providers](https://www.entropywins.wtf/blog/2017/10/09/yield-in-phpunit-data-providers/) →

[\(https://www.entropywins.wtf/blog/2017/10/09/yield-in-phpunit-data-providers/\)](https://www.entropywins.wtf/blog/2017/10/09/yield-in-phpunit-data-providers/)

13 thoughts on “The Fallacy of DRY”



Karl Metivier says:

2017-10-30 at 05:09 (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comment-354702>)

If you follow the 4 rules of Simple Design (from Kent Beck's XP Programming), you probably will not succumb to the fallacy of DRY:

- 1-Passes all the tests.
 - 2-Express every idea we need to express.
 - 3-Contains no duplication. (DRY)
 - 4- Minimized the number of classes, methods and other moving parts. (YAGNI)
- (in that order !)

Reply →



Mikko Rantalainen (<https://plus.google.com/+MikkoRantalainen>) says:
2018-01-10 at 11:42 (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comment-354721>)

I think you should swap points 3 and 4. Minimizing the total amount of information is more important than avoiding copy-paste code (always add a comment that mentions other location(s) for the same or very similar code so that future modifications can be done to both locations).

Reply →



Matheus Jahnke says:

2022-03-06 at 02:58 (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comment-355969>)

Wouldn't minimizing the number of classes and methods lead to a big, oversized, single main function?

> always add a comment that mentions other location(s) for the same or very similar code so that future modifications can be done to both locations

This is something I would disagree quite strongly, and also helps me make a point: things things that happen identical might not be duplicates, 1 might be the magic number, the index of the first valid element in a list(where the 0th element is some dummy), or the neutral element of multiplication(if you want to compute the product of the elements in a list)

declaring a variable called one, using it instead every time instead of using a literal number 1 is ridiculous(and note that "one" has 3 chars while "1" has... 1), but if a sequence of appearances of 1 has the same meaning(a.k.a. you have to change them all if you change one

of the), then it might be useful to make a variable(or, even better if it is the case, a constant) so it explicitly expresses that idea(especially if it's a magic number, or a very specific index). Note that this is made so the specific idea is better expressed(a.k.a. is it some magic number? is it an index?)

TLDR: I'd say it DRY done right(tm) relies on "Express every idea we need to express." because I've the (spooky) bad duplication is a result of a badly expressed idea.

Of course, the "reduction" must improve the way the idea is expressed. I believe DRY fails when people choose the wrong abstraction(or treat some thing as duplication when it actually isn't)
<https://sandimetz.com/blog/2016/1/20/the-wrong-abstraction>
(<https://sandimetz.com/blog/2016/1/20/the-wrong-abstraction>)

Reply →

Pingback: Why Every Single Argument of Dan North is Wrong – Entropy Wins
(<https://www.entropywins.wtf/blog/2017/02/17/why-every-single-argument-of-dan-north-is-wrong/>).

Pingback: Generic Entity handling code – Entropy Wins
(<https://www.entropywins.wtf/blog/2017/05/24/generic-entity-handling-code/>).

Pingback: Clean Architecture: UseCase tests – Entropy Wins
(<https://www.entropywins.wtf/blog/2018/08/01/clean-architecture-usecase-tests/>).

Pingback: Readable Functions: Minimize State – Entropy Wins
(<https://www.entropywins.wtf/blog/2018/10/24/readable-functions-minimize-state/>).

Pingback: PHP Typed Properties – Entropy Wins
(<https://www.entropywins.wtf/blog/2018/10/28/php-typed-properties/>).



chasepeeler says:

2021-02-12 at 20:22 (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comment-355700>)

There are MANY times where a rather straight-forward block of code with a few repetitions becomes much more convoluted if I you try and refactor it to remove the repetitions. I'm a big proponent of DRY, as I think more often than not it's a good rule of thumb, but like everything, it's a guideline, not a hard rule.

[Reply](#) →



Matheus Jahnke says:

2022-03-06 at 03:09 (<https://www.entropywins.wtf/blog/2017/09/06/the-fallacy-of-dry/#comment-355970>)

Also, on the 2 examples:

```
for($i=0; $i<4, ++$i) {  
  doAction($i);  
}
```

vs

```
doAction(1);  
doAction(2);  
doAction(3);
```

What about using something like this:

```
(Javascript)  
[1,2,3].forEach(doAction)
```

```
(Python)  
for i in [1,2,3]:  
  doAction(i)
```

```
(Haskell)  
forM_ [1,2,3] doAction
```

(Or even)

forM_ [1..3] doAction


The first 2 might be intuitive, the last one in haskell is might require some background, but essentially:

for every element in the list[1,2,3], in the order that they appear, execute an action with element as input;

Ignore return value(if you wanted the return value, you would use forM, without an underscore)

The impression I'm getting is that, sometimes, DRY makes things less explicit(which might not be a problem... until it is)

Reply →

Pingback: [Advice for Junior Developers - DEV Community](https://dev.to/jeroendedauw/advice-for-junior-developers-30am) 
(<https://dev.to/jeroendedauw/advice-for-junior-developers-30am>)

Pingback: [Advice for junior developers - Entropy Wins](https://www.entropywins.wtf/blog/2022/09/21/advice-for-junior-developers/)
(<https://www.entropywins.wtf/blog/2022/09/21/advice-for-junior-developers/>)

LEAVE A REPLY

Enter your comment here...

This site uses Akismet to reduce spam. [Learn how your comment data is processed \(https://akismet.com/privacy/\)](https://akismet.com/privacy/).

Search ...

START YOUR WIKI TODAY

Try out [ProWiki \(https://www.pro.wiki/\)](https://www.pro.wiki/) for free. Managed [MediaWiki hosting \(https://www.pro.wiki/\)](https://www.pro.wiki/) with one-click install and admin panel. [Create your wiki today \(https://www.pro.wiki/\)](https://www.pro.wiki/).

NEWSLETTER

Sign up below to receive news on my upcoming Clean Architecture book, including a discount:

Your email address

SUBSCRIBE

RECENT POSTS

[Advice for junior developers \(https://www.entropywins.wtf/blog/2022/09/21/advice-for-junior-developers/\)](https://www.entropywins.wtf/blog/2022/09/21/advice-for-junior-developers/)

[Using PSR-3 Monolog in MediaWiki \(https://www.entropywins.wtf/blog/2022/02/08/using-psr3-monolog-in-mediawiki/\)](https://www.entropywins.wtf/blog/2022/02/08/using-psr3-monolog-in-mediawiki/)

[Value Objects with PHP 8.1 \(https://www.entropywins.wtf/blog/2022/01/30/value-objects-with-php-8-1/\)](https://www.entropywins.wtf/blog/2022/01/30/value-objects-with-php-8-1/)

[New MediaWiki blog \(https://www.entropywins.wtf/blog/2019/10/26/new-mediawiki-blog/\)](https://www.entropywins.wtf/blog/2019/10/26/new-mediawiki-blog/)

[Applications as Frameworks \(https://www.entropywins.wtf/blog/2019/02/28/applications-as-frameworks/\)](https://www.entropywins.wtf/blog/2019/02/28/applications-as-frameworks/)

[Readable Functions: Guard Clause \(https://www.entropywins.wtf/blog/2019/01/14/readable-functions-guard-clause/\)](https://www.entropywins.wtf/blog/2019/01/14/readable-functions-guard-clause/)

[My year in books \(https://www.entropywins.wtf/blog/2019/01/03/my-year-in-books-4/\)](https://www.entropywins.wtf/blog/2019/01/03/my-year-in-books-4/)

[Readable Functions: Do One Thing \(https://www.entropywins.wtf/blog/2018/10/30/readable-functions-do-one-thing/\)](https://www.entropywins.wtf/blog/2018/10/30/readable-functions-do-one-thing/)

[PHP Typed Properties \(https://www.entropywins.wtf/blog/2018/10/28/php-typed-properties/\)](https://www.entropywins.wtf/blog/2018/10/28/php-typed-properties/)

[Readable Functions: Minimize State \(https://www.entropywins.wtf/blog/2018/10/24/readable-functions-minimize-state/\)](https://www.entropywins.wtf/blog/2018/10/24/readable-functions-minimize-state/)

ARCHIVES

[September 2022 \(https://www.entropywins.wtf/blog/2022/09/\)](https://www.entropywins.wtf/blog/2022/09/)

[February 2022 \(https://www.entropywins.wtf/blog/2022/02/\)](https://www.entropywins.wtf/blog/2022/02/)

[January 2022 \(https://www.entropywins.wtf/blog/2022/01/\)](https://www.entropywins.wtf/blog/2022/01/)

[October 2019 \(https://www.entropywins.wtf/blog/2019/10/\)](https://www.entropywins.wtf/blog/2019/10/)

[February 2019 \(https://www.entropywins.wtf/blog/2019/02/\)](https://www.entropywins.wtf/blog/2019/02/)

[January 2019 \(https://www.entropywins.wtf/blog/2019/01/\)](https://www.entropywins.wtf/blog/2019/01/)

[October 2018 \(https://www.entropywins.wtf/blog/2018/10/\)](https://www.entropywins.wtf/blog/2018/10/)

[September 2018 \(https://www.entropywins.wtf/blog/2018/09/\)](https://www.entropywins.wtf/blog/2018/09/)

[August 2018 \(https://www.entropywins.wtf/blog/2018/08/\)](https://www.entropywins.wtf/blog/2018/08/)

[May 2018 \(https://www.entropywins.wtf/blog/2018/05/\)](https://www.entropywins.wtf/blog/2018/05/)

[February 2018 \(https://www.entropywins.wtf/blog/2018/02/\)](https://www.entropywins.wtf/blog/2018/02/)

[January 2018 \(https://www.entropywins.wtf/blog/2018/01/\)](https://www.entropywins.wtf/blog/2018/01/)

[October 2017 \(https://www.entropywins.wtf/blog/2017/10/\)](https://www.entropywins.wtf/blog/2017/10/)

[September 2017 \(https://www.entropywins.wtf/blog/2017/09/\)](https://www.entropywins.wtf/blog/2017/09/)

[June 2017 \(https://www.entropywins.wtf/blog/2017/06/\)](https://www.entropywins.wtf/blog/2017/06/)

[May 2017 \(https://www.entropywins.wtf/blog/2017/05/\)](https://www.entropywins.wtf/blog/2017/05/)

[April 2017 \(https://www.entropywins.wtf/blog/2017/04/\)](https://www.entropywins.wtf/blog/2017/04/)

[February 2017 \(https://www.entropywins.wtf/blog/2017/02/\)](https://www.entropywins.wtf/blog/2017/02/)

[January 2017 \(https://www.entropywins.wtf/blog/2017/01/\)](https://www.entropywins.wtf/blog/2017/01/)

[December 2016 \(https://www.entropywins.wtf/blog/2016/12/\)](https://www.entropywins.wtf/blog/2016/12/)

[November 2016 \(https://www.entropywins.wtf/blog/2016/11/\)](https://www.entropywins.wtf/blog/2016/11/)

[October 2016 \(https://www.entropywins.wtf/blog/2016/10/\)](https://www.entropywins.wtf/blog/2016/10/)

[September 2016 \(https://www.entropywins.wtf/blog/2016/09/\)](https://www.entropywins.wtf/blog/2016/09/)

[August 2016 \(https://www.entropywins.wtf/blog/2016/08/\)](https://www.entropywins.wtf/blog/2016/08/)

[July 2016 \(https://www.entropywins.wtf/blog/2016/07/\)](https://www.entropywins.wtf/blog/2016/07/)

[June 2016 \(https://www.entropywins.wtf/blog/2016/06/\)](https://www.entropywins.wtf/blog/2016/06/)

[May 2016 \(https://www.entropywins.wtf/blog/2016/05/\)](https://www.entropywins.wtf/blog/2016/05/)

[April 2016 \(https://www.entropywins.wtf/blog/2016/04/\)](https://www.entropywins.wtf/blog/2016/04/)

[March 2016 \(https://www.entropywins.wtf/blog/2016/03/\)](https://www.entropywins.wtf/blog/2016/03/)

[February 2016 \(https://www.entropywins.wtf/blog/2016/02/\)](https://www.entropywins.wtf/blog/2016/02/)

[January 2016 \(https://www.entropywins.wtf/blog/2016/01/\)](https://www.entropywins.wtf/blog/2016/01/)

[December 2015 \(https://www.entropywins.wtf/blog/2015/12/\)](https://www.entropywins.wtf/blog/2015/12/)

[November 2015 \(https://www.entropywins.wtf/blog/2015/11/\)](https://www.entropywins.wtf/blog/2015/11/)

[October 2015 \(https://www.entropywins.wtf/blog/2015/10/\)](https://www.entropywins.wtf/blog/2015/10/)

[September 2015 \(https://www.entropywins.wtf/blog/2015/09/\)](https://www.entropywins.wtf/blog/2015/09/)

[August 2015 \(https://www.entropywins.wtf/blog/2015/08/\)](https://www.entropywins.wtf/blog/2015/08/)

[June 2015 \(https://www.entropywins.wtf/blog/2015/06/\)](https://www.entropywins.wtf/blog/2015/06/)

[February 2015 \(https://www.entropywins.wtf/blog/2015/02/\)](https://www.entropywins.wtf/blog/2015/02/)

[September 2014 \(https://www.entropywins.wtf/blog/2014/09/\)](https://www.entropywins.wtf/blog/2014/09/)

[August 2014 \(https://www.entropywins.wtf/blog/2014/08/\)](https://www.entropywins.wtf/blog/2014/08/)

[July 2014 \(https://www.entropywins.wtf/blog/2014/07/\)](https://www.entropywins.wtf/blog/2014/07/)

[May 2014 \(https://www.entropywins.wtf/blog/2014/05/\)](https://www.entropywins.wtf/blog/2014/05/)

[April 2014 \(https://www.entropywins.wtf/blog/2014/04/\)](https://www.entropywins.wtf/blog/2014/04/)

[March 2014 \(https://www.entropywins.wtf/blog/2014/03/\)](https://www.entropywins.wtf/blog/2014/03/)

[February 2014 \(https://www.entropywins.wtf/blog/2014/02/\)](https://www.entropywins.wtf/blog/2014/02/)

[January 2014 \(https://www.entropywins.wtf/blog/2014/01/\)](https://www.entropywins.wtf/blog/2014/01/)

[December 2013 \(https://www.entropywins.wtf/blog/2013/12/\)](https://www.entropywins.wtf/blog/2013/12/)

[November 2013 \(https://www.entropywins.wtf/blog/2013/11/\)](https://www.entropywins.wtf/blog/2013/11/)

[October 2013 \(https://www.entropywins.wtf/blog/2013/10/\)](https://www.entropywins.wtf/blog/2013/10/)

[September 2013 \(https://www.entropywins.wtf/blog/2013/09/\)](https://www.entropywins.wtf/blog/2013/09/)

[July 2013 \(https://www.entropywins.wtf/blog/2013/07/\)](https://www.entropywins.wtf/blog/2013/07/)

[June 2013 \(https://www.entropywins.wtf/blog/2013/06/\)](https://www.entropywins.wtf/blog/2013/06/)

[March 2012 \(https://www.entropywins.wtf/blog/2012/03/\)](https://www.entropywins.wtf/blog/2012/03/)

[December 2011 \(https://www.entropywins.wtf/blog/2011/12/\)](https://www.entropywins.wtf/blog/2011/12/)

[November 2011 \(https://www.entropywins.wtf/blog/2011/11/\)](https://www.entropywins.wtf/blog/2011/11/)

[September 2011 \(https://www.entropywins.wtf/blog/2011/09/\)](https://www.entropywins.wtf/blog/2011/09/)

[August 2011 \(https://www.entropywins.wtf/blog/2011/08/\)](https://www.entropywins.wtf/blog/2011/08/)

[July 2011 \(https://www.entropywins.wtf/blog/2011/07/\)](https://www.entropywins.wtf/blog/2011/07/)

[June 2011 \(https://www.entropywins.wtf/blog/2011/06/\)](https://www.entropywins.wtf/blog/2011/06/)

[May 2011 \(https://www.entropywins.wtf/blog/2011/05/\)](https://www.entropywins.wtf/blog/2011/05/)

[February 2011 \(https://www.entropywins.wtf/blog/2011/02/\)](https://www.entropywins.wtf/blog/2011/02/)

[January 2011 \(https://www.entropywins.wtf/blog/2011/01/\)](https://www.entropywins.wtf/blog/2011/01/)

[December 2010 \(https://www.entropywins.wtf/blog/2010/12/\)](https://www.entropywins.wtf/blog/2010/12/)

[October 2010 \(https://www.entropywins.wtf/blog/2010/10/\)](https://www.entropywins.wtf/blog/2010/10/)

[August 2010 \(https://www.entropywins.wtf/blog/2010/08/\)](https://www.entropywins.wtf/blog/2010/08/)

[July 2010 \(https://www.entropywins.wtf/blog/2010/07/\)](https://www.entropywins.wtf/blog/2010/07/)

[June 2010 \(https://www.entropywins.wtf/blog/2010/06/\)](https://www.entropywins.wtf/blog/2010/06/)

[May 2010 \(https://www.entropywins.wtf/blog/2010/05/\)](https://www.entropywins.wtf/blog/2010/05/)

[April 2010 \(https://www.entropywins.wtf/blog/2010/04/\)](https://www.entropywins.wtf/blog/2010/04/)

[March 2010 \(https://www.entropywins.wtf/blog/2010/03/\)](https://www.entropywins.wtf/blog/2010/03/)

[February 2010 \(https://www.entropywins.wtf/blog/2010/02/\)](https://www.entropywins.wtf/blog/2010/02/)

[January 2010 \(https://www.entropywins.wtf/blog/2010/01/\)](https://www.entropywins.wtf/blog/2010/01/)

[December 2009 \(https://www.entropywins.wtf/blog/2009/12/\)](https://www.entropywins.wtf/blog/2009/12/)

[November 2009 \(https://www.entropywins.wtf/blog/2009/11/\)](https://www.entropywins.wtf/blog/2009/11/)

[October 2009 \(https://www.entropywins.wtf/blog/2009/10/\)](https://www.entropywins.wtf/blog/2009/10/)

[September 2009 \(https://www.entropywins.wtf/blog/2009/09/\)](https://www.entropywins.wtf/blog/2009/09/)

[August 2009 \(https://www.entropywins.wtf/blog/2009/08/\)](https://www.entropywins.wtf/blog/2009/08/)

[July 2009 \(https://www.entropywins.wtf/blog/2009/07/\)](https://www.entropywins.wtf/blog/2009/07/)

[June 2009 \(https://www.entropywins.wtf/blog/2009/06/\)](https://www.entropywins.wtf/blog/2009/06/)

[May 2009 \(https://www.entropywins.wtf/blog/2009/05/\)](https://www.entropywins.wtf/blog/2009/05/)

[April 2009 \(https://www.entropywins.wtf/blog/2009/04/\)](https://www.entropywins.wtf/blog/2009/04/)

Copyright © 2009-2020 Jeroen De Dauw