

Inspired Web Development

(<https://www.timsommer.be>)



DEVELOPMENT (/TAG/DEVELOPMENT/)

ARCHITECTURE (/TAG/ARCHITECTURE/)

LAW (/TAG/LAW/)

# Famous Laws Of Software Development

December 18, 2017



(/content/images/2017/12/giammarco-boscaro-380903-1.jpg)

Like any other field, the world of Software Development has some interesting and famous rules, principles and laws. Programmers, developers, managers and architects often use these in conversations, meetings and chats. More than often we tend to nod along, not willing to let our conversation partner know we've actually never heard of these Brook, Moore or Wirth characters.

These laws consist of rules, principles, or famous words from great and inspiring persons in the development world. At the same time they are interesting, funny, worth knowing, and all have great back-stories which are amazing to read.

In this post I am going to share my collection, interpretation and thoughts on the most famous and most used laws in Software Development.

This is kind-off a big post, so I've included an index if you want to skip some:

- Murphy's Law
- Brook's Law
- Hofstadter's Law
- Conway's Law

- Postel's Law
- Pareto Principle
- The Peter Principle
- Kerckhoffs's Principle
- Linus's Law
- Moore's Law
- Wirth's law
- Ninety-ninety rule
- Knuth's optimization principle
- Norvig's Law

## Murphy's Law

Probably one of the most famous of all laws, mostly because it is not only applicable to Software Development.

If something can go wrong, it will.

**First derivation:** If it works, you probably didn't write it.

**Second derivation:** Cursing is the only language all programmers speak fluently.

**Conclusion:** A computer will do what you write, not what you want.

Defensive programming, version control, doom scenario's (for those damned zombie-server-attacks), TDD, MDD, etc. are all are good practices for defending against this law.

## Brook's Law

Most developers will -either knowingly or unknowingly- have experience with Brook's law, which states:

Adding manpower to a late software project makes it later.

If a project is running late, simply adding manpower will most likely have disastrous results. Looking and reviewing the level of programming efficiency, the software methodology, the technical architecture, etc. will almost always have better outcomes. Or not, which probably

means Hofstadter's Law is also in play.

## Hofstadter's Law



The Hofstadter's law was written by Douglas Hofstadter and named after him.

This law is -off course- not to be confused with Leonard Hofstadter from the Big Bang theory. Even though his quotation will make sense to some you.

The rule states:

It always takes longer than you expect, even when you take into account Hofstadter's Law.

The "law" is a statement regarding the difficulty of accurately estimating the time it will take to complete tasks of substantial complexity. The recursive nature of the law is a reflection of the widely experienced difficulty of estimating complex tasks despite all best efforts, including knowing that the task is complex.

That's why you must always have a buffer before you give any sort of estimation. If you want to know more on how to provide better estimations, read my post on the subject: Estimation Wizardry (/estimation-wizardry).

## Conway's Law

Any piece of software reflects the organizational structure that produced it.

Or even more clearly:

Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.

In many organizations teams are divided according to their functional skills. So you'll have teams of front-end developers, backend-developers and database developers. Simply stated, it's hard for someone to change something if the thing he/she wants to change is owned by someone else.

It is much better, and more and more implemented as such, to deploy teams around a bounded context. Architectures such as microservices structure their teams around service boundaries rather than siloed technical architecture partitions.

So, structure teams to look like your target architecture, and it will be easier to achieve it. That's how you defend against Conways's law.

## Postel's Law aka Robustness principle

Be conservative in what you send, be liberal in what you accept

Jon Postel originally articulated this as a principle for making TCP implementations robust. This principle is also embodied by HTML which many attribute as a cause of its success and failure, depending on who you ask.

In today's highly charged political environment, Postel's law is a uniter.

## Pareto Principle aka The 80-20 rule

For many phenomena, 80% of consequences stem from 20% of the causes.

This is the principle behind the painful truth that 80% of the bugs in the code arise from 20% of the code.

Otherwise stated, 80% of the work done in a company is performed by 20% of the staff. The problem is you don't always have a clear idea of which 20%.

## The Peter Principle

A pretty depressing and at times frustrating law, certainly if you happen to have firsthand experience.

In a hierarchy, every employee tends to rise to his level of incompetence.

Just read Dilbert (or watch The Office) to get some examples of this in action.  
As for Dilbert, this one is far out my favorite!



## Kerckhoffs's Principle

In cryptography, a system should be secure even if everything about the system, except for a small piece of information - the key - is public knowledge.

This is the main principle underlying public key cryptography.

## Linus's Law

Named after Linus Torvalds, the creator of Linux, this law states:

Given enough eyeballs, all bugs are shallow.

This law was described using the famous The Cathedral and the Bazaar ([https://en.wikipedia.org/wiki/The\\_Cathedral\\_and\\_the\\_Bazaar](https://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar)) essay, explaining the contrast between two different free software development models:

- The Cathedral model, in which source code is available with each software release, but code developed between releases is restricted to an exclusive group of software developers.
- The Bazaar model, in which the code is developed over the **Internet in view of the public**.

The conclusion? The more widely available the source code is for public testing, scrutiny, and experimentation, the more rapidly all forms of bugs will be discovered.

## Moore's Law

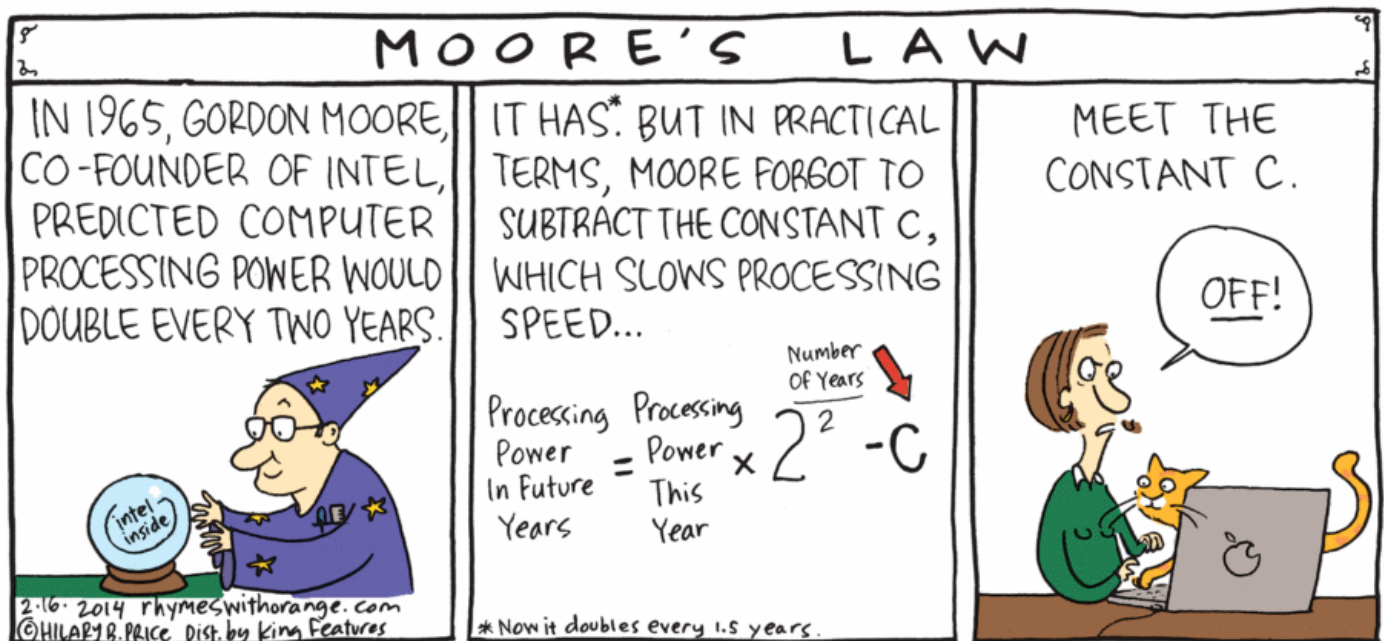
The power of computers per unit cost doubles every 24 month.

The most popular version states:

The number of transistors on an integrated circuit will double in about 18 months.

Or

The processing speed of computers will double every two years!



## Wirth's law

Software gets slower faster than hardware gets faster.

Take that Moore's Law!

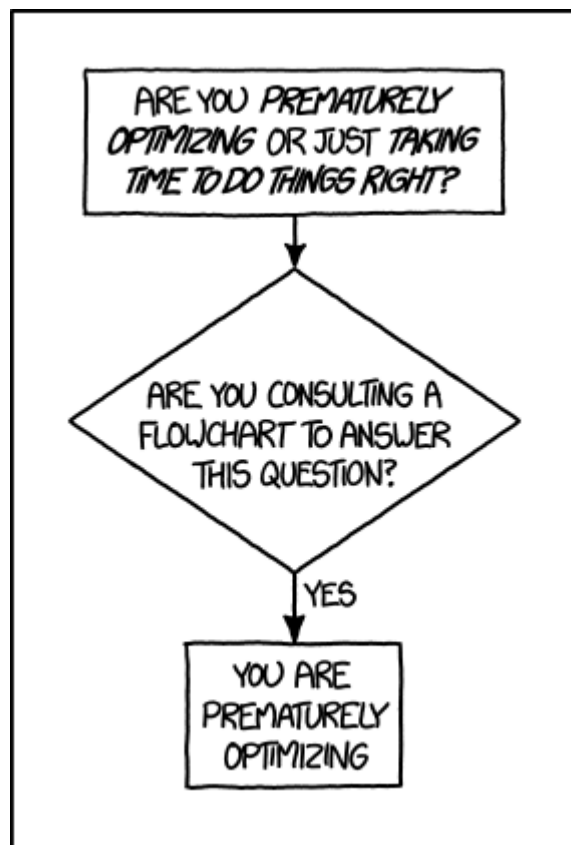
## Ninety-ninety rule

The first 90% of the code takes 10% of the time. The remaining 10% takes the other 90% of the time

So true. Anyone who does not agree with this?

## Knuth's optimization principle

Premature optimization is the root of all evil.



First you write code, then you identify bottlenecks, then you fix!

## Norvig's Law

Any technology that surpasses 50% penetration will never double again (in any number of months).

## Conclusion

I love these 'laws' as they each have their own story, their own background. And as a consultant or architect you have to familiarize yourself with them. So for me, and possibly for you, I know have a list of the most used and most famous laws in Software Development!




Did I miss your favorite law? Do (or don't) you agree with each law, or have any (funny) experience with one of them? What is your favorite? Let me know using the comment section!


---


Share on

 ([http://twitter.com/share?](http://twitter.com/share?text=Famous%20laws%20of%20Software%20development&url=https://www.timsommer.be/famous-laws-of-software-development/)

[text=Famous%20laws%20of%20Software%20development&url=https://www.timsommer.be/famous-laws-of-software-development/](http://twitter.com/share?text=Famous%20laws%20of%20Software%20development&url=https://www.timsommer.be/famous-laws-of-software-development/))

 (<https://www.facebook.com/sharer/sharer.php?u=https://www.timsommer.be/famous-laws-of-software-development/>)

 (<https://plus.google.com/share?url=https://www.timsommer.be/famous-laws-of-software-development/>)

 (<https://pinterest.com/share?url=https://www.timsommer.be/famous-laws-of-software-development/>)

[Newer Articles \(/2017-in-praise-of-idleness/\)](/2017-in-praise-of-idleness/)

[Older Articles \(/an-introduction-to-deployment-pipelines/\)](/an-introduction-to-deployment-pipelines/)



**Tim Sommer** (</author/tim/>)

I'm a Web Developer and Windows Insider MVP with +8 years of experience in the .NET framework, with a strong focus on HTML5 and JavaScript development.

 (<https://www.facebook.com/s0mmerTim>)

 (<https://twitter.com/sommertim>)

 (<https://blog.timsommer.be/rss>)

**44 Comments**

**Tim Sommer - Inspired Web Development**

 **Login** ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS **Corey Vayette** • 3 months ago

I'm shocked Cunningham's law isn't on this list.

- Cunningham's Law: The fastest way to get help over the internet is not to ask the question but to instead answer it wrong.

13 ^ | v • Reply • Share ›

**Astrapto** → **Corey Vayette** • 3 months ago

Maybe its omission was a conscious choice to invoke it :)

1 ^ | v • Reply • Share ›

**Guy Srinivasan** • 3 months ago

Goodhart's Law: When a measure becomes a target, it ceases to be a good measure.

For just one of many examples, code coverage statistics.

7 ^ | v • Reply • Share ›

**Michael G Alcock** → **Guy Srinivasan** • 3 months ago

And don't forget the classic: " constants aren't and variables won't"

^ | v • Reply • Share ›

**Karrq** → **Michael G Alcock** • 3 months ago

Rust wishes to discuss this with you

^ | v • Reply • Share ›

**kgu87** • 3 months ago

Parkinson's Law is sorely missing from this list. "Work expands to fill all time allocated to its completion".

4 ^ | v • Reply • Share ›

**Rudi Larno** • 3 months ago

Might as well throw in <https://en.wikiquote.org/wi...>

3 ^ | v • Reply • Share ›

**Christian “kiko” Reis** • 3 months ago

You missed my infamous law: "software never works the first time", and its corollary "make sure the first time you use new software isn't production-critical".

3 ^ | v • Reply • Share ›

**Andrew Powers** → **Christian “kiko” Reis** • 3 months ago



Good one.

4 ^ | v • Reply • Share ›

**yebyen** • 3 months ago

Don't forget Gall's Law:

A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system.

1 ^ | v • Reply • Share ›

**Yogthos** • 3 months agoAlso Koomey's law [https://en.wikipedia.org/wiki...](https://en.wikipedia.org/wiki/Koomey's_law)

1 ^ | v • Reply • Share ›

**Svendrik Frinlan** • 3 months ago

While it doesn't go to the broader level of some of these and is not a "law", Greenspun's Tenth Rule states:

"Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp."

- A quote saying that despite attempting to write in a simpler or lower level language, you're going to end up building out features of higher languages when projects become complex enough. However, since you're writing it on the fly, you're likely to miss out on the edge cases and guards that intentional higher level languages would have.

There's also Zawinski's law:

"Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can."

- A commentary on feature creep of even the most specific, purpose driven apps.

Bonus quote attributed to Zawinski:

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems."

- You can replace regular expressions with any powerful tool that is easy to start using, but difficult to use correctly.

1 ^ | v • Reply • Share ›

**bigblind1991** • 3 months ago

I'd like to add Spolsky's law:

■ All non-trivial abstractions, to some degree, are leaky.

1 ^ | v • Reply • Share ›

**Doug** ➔ **bigblind1991** • 3 months ago

I wish I could remember the author, but I saw a great quote about fifteen years ago. Basically, in C, you write a specific solution to a problem. In C++ you write an approximation of a solution to a problem, that also happens to be a rough approximation to a solution to several other problems.

^ | v • Reply • Share ›

**Hillbilly** • 2 months ago

**timbony** · 2 months ago

You misquoted the 90 percent law, it should be:

The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.

^ | v · Reply · Share ›

**Nicolas Iensen** · 3 months ago

Metcalfe's law says that the effect of a telecommunications network is proportional to the square of the number of connected users of the system.

I like to use this law to explain why teams with too many members have trouble to communicate efficiently.

^ | v · Reply · Share ›

**lorg** · 3 months ago

Re Conway's law: I also heard it said the following way: the number of modules in given system will be the number of developers/teams developing it.

^ | v · Reply · Share ›

**Wade Mealing** · 3 months ago

I strongly believe that Postel's Law is incorrect. You should be -very- conservative in what you accept, accepting liberally is a path to madness and guaranteed to give you headaches and pain in the long term. You can always be "MORE" liberal in what you accept at a later date, pulling back from what you accept at a later date is a much harder problem. You're likely to find that something, somewhere is misusing your system in a way you'd never expect and you've just upset them.

^ | v · Reply · Share ›

**Ja Du** · 3 months ago

Thanks for doing this, interesting!

Hofstadter's Law is cool and one I run into always. He's written a lot of cool stuff on recursion and strange loops, which make's it very apropos that his law is recursive.

I also like one that's not mentioned, The Law of Demeter which I've found very butt-saving when refactoring. You could stock many a volume with all the Laws that are out there, too bad there's no authority that can issue citations when people don't pay attention to them. :)

^ | v · Reply · Share ›

**John Spooner** · 3 months ago

I have often found that no temporary software project or fix ever is. There is no law I have seen supporting this. It is just my humble observation.

^ | v · Reply · Share ›

**Chuck Marshall** → **John Spooner** · 3 months ago

The lifetime of any system is inversely proportional to its intended lifetime.

1 ^ | v · Reply · Share ›

**mikedux** · 3 months ago

I will argue that Murphy's Law, as listed, is actually **Finagle's Law of Dynamic Negatives**  
"Anything that can go wrong, will. "

While **Murphy's Law** is "If there are two or more ways to do something, and one of those ways can result in a catastrophe, then someone will do it."

And of course we must include O'Toole's Corollary of Finagle's Law - "The perversity of the Universe tends towards a maximum."

And And **Muphry's Law** – "If you write anything criticizing editing or proofreading, there will be a fault of some kind in what you have written."

Any code of your own that you haven't looked at for six or more months might as well have been written by someone else. -**Eagleson's Law**

Adding manpower to a late software project makes is later. -**Brooke's Law**

The speed of software halves every 18 months. - **Gates's Law** In reference to Moore's Law

Data expands to fill the space available for storage. - **Parkinson's Law of Data**

Ninety percent of everything is crud. **Sturgeon's Law**

The square root of the number of people on a project write 50% of the code. - **Angulation of Price's Law**

**Parkinson's Corollary:** "Expenditure rises to meet income."

^ | v • Reply • Share ›



**Doug** • 3 months ago

Doug's Law. (yeah, its mine...) All software wears out with age. Eventually you have to start over with a clean slate.

^ | v • Reply • Share ›



**Ilya Zub** • 3 months ago

Law of the instrument (Maslow's hammer): I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.

^ | v • Reply • Share ›



**Sven Åge Jørgensen** → Ilya Zub • 3 months ago

Teh law of my father "Laurits law": Never use force, simply use a larger hammer!

^ | v • Reply • Share ›



**Christian Lynbech** • 3 months ago

Poels rule: Allow none of foo, one of foo, or any number of foo.

<https://en.wikipedia.org/wi...>

^ | v • Reply • Share ›



**Christian Lynbech** • 3 months ago

Ok, it is a bit of paraphrase, but I am very fond of this (my suggestion of naming):

The Post rule: Real Programmers program in FORTRAN and can do so in any language.

<https://www.ee.ryerson.ca/~...>

^ | v • Reply • Share ›



**Christian Lynbech** • 3 months ago

Amara's Law: we tend to overestimate the impact of a new technology in the short run, but we underestimate it in the long run.

<http://www.rationaloptimist...>

^ | v • Reply • Share ›



**Christian Lynbech** • 3 months ago

As also described on the wikipedia page (<https://en.wikipedia.org/wi...>, the law that Moore stated in his article, was specifically about number of transistors, that is not just a popular variation. In fact, as processing power goes, he had to cheat (as I remember it) as the final dot confirming the doubling was a RAM chip and not a CPU! While CPU processing power has indeed increased enormously, the definition of CPU power or speed is only vaguely defined and its connection to the number of transistors is even harder to quantify.

^ | v • Reply • Share ›



**Guy Gordon** • 3 months ago

My favorite: "A programmer has a problem. He decides to solve it using Regular Expressions." ... (You know how it goes.)

^ | v • Reply • Share ›



**Guy Gordon** → **Guy Gordon** • 3 months ago

Yes, of course it's a Law.

^ | v • Reply • Share ›



**Dan D.** • 3 months ago

It's worth explaining the Peter Principle a bit better, especially as Dilbert and The Office never actually show the Peter Principle in action, but rather its consequences.

In particular, it's the fact that anyone who is competent at their job will be continually promoted. This is a good thing. The bad part of it is that eventually, they'll be competent at something and then promoted to something they're incompetent at, which is where they'll stay. In contrast, it's hard to imagine that the PHB or Michael Scott (to a lesser extent) were ever competent at anything they did before.

One thing that Dilbert does illustrate occasionally is the similar but related "Failing Upwards", which is where someone gets promoted despite incompetence to get them out of someone else's chain of command, and is sometimes confused for the Peter Principle.

^ | v • Reply • Share ›



**Ramon Georges** → **Dan D.** • 3 months ago

I often associate the Peter Principle with the Dunning Kruger Effect.

^ | v • Reply • Share ›



**Doug** → **Dan D.** • 3 months ago

In my thirty-five years in this industry I have seen two people 'fail upward' in spectacular fashion.

^ | v • Reply • Share ›



**Greg Torking** • 3 months ago

**Greg Iomkins** • 3 months ago

I can't think of a pithy summary but in my many years of replacing old systems with relatively modern ones:

From a users point of view, the best way or the cheapest way or the most efficient way is of no interest. All that really matters is the way the old system did it, and not changing that.

^ | v • Reply • Share ›

**fzammetti** • 3 months ago

I've never (completely) agreed with Knuth's optimization principle.

The extent to which I agree is certainly that you need working code before you can have optimal code, taken as a generality. No argument from me that getting the code to work before you worry much about how it performs is the right approach.

But, I think that developers are often times too rigid with that thinking. What I mean is that if you have habits that AUTOMATICALLY result in more optimized code then it's no longer "premature" optimization, it's simply writing better code by default.

As a simple example, take:

```
for (int i = 0; i < someArray.length; i++) { }
```

Developers recognize, assuming there's no compiler to optimize it for you, that you shouldn't access length with every iteration and should instead do:

```
int arrayLength = someArray.length;  
for (int i = 0; i < arrayLength; i++) { }
```

---

see more

^ | v • Reply • Share ›

**Didi Bear** → fzammetti • 3 months ago

Micro-optimisations are a wound for newcomers into the code and they solve nothing compare to the framework they run on.

In your example, store the number of iteration is a common usage, mostly because it is done in low level language. But if I used memoization in every functions, I'm not sure my colleagues still like me.

Micro-optimisation are ok on low level language used for example in micro-controller, in which memory is important.

^ | v • Reply • Share ›

**Berithjah** → fzammetti • 3 months ago

I disagree completely. Optimisations that don't actually end up doing anything meaningful and are also bad aren't worth it. Even if you add them all up, all those zeros are still zero. In fact, I'd argue this is actually a negative though, since you are arguing for introducing a local variable in this case (which pollutes the function) and is not constant/tied to the array within the code anymore. Also you aren't arguing with benchmarks which is core for whenever discussing the utility of optimisations, so obviously you don't understand Knuth's whole schpiel on optimisation work, you don't

understand it. Sorry to come off so negative, but I encourage you to whip out a profiler and have some fun with the JVM. It's a lot more fulfilling and knowledge expanding than off-the-cuff remarks about "loop optimizations", which will make code reviews much more meaningful.

Why is this a complete non-optimisation?

1) I assume you are talking about Java? Java does run time array bound checks. If you access the *i*th index of an array, Java actually has some boiler plate code to prevent access to negative or too large values. Why? Because you are programming in Java and it runs a "virtual machine". So the length is being read from memory regardless.

[see more](#)

^ | v • Reply • Share ›



**Cole Calistra** • 3 months ago

Forgot Vogels' Law: "Everything breaks, all the time" - Dr. Werner Vogels - CTO Amazon.com

^ | v • Reply • Share ›



**Kurtis Miller** • 3 months ago

The Ninety-Ninety rule is stated differently (and less effectively) than I've always heard it:

The first 90 percent of the code accounts for the first 90 percent of the development time.  
The remaining 10 percent of the code accounts for the other 90 percent of the development time.

(Credited to Tom Cargill, Bell Labs)

^ | v • Reply • Share ›



**Ingmar Heinrich** • 3 months ago

I don't get Norvig's Law:

Any technology that surpasses 50% penetration will never double again (in any number of months).

This seems trivial. Nothing above 50% can double, because then it would be more than 100%.

^ | v • Reply • Share ›



**iBadger** ➔ Ingmar Heinrich • 3 months ago

It is trivial, but yet you'd be amazed how often it is neglected or overlooked.

2 ^ | v • Reply • Share ›



**gavingreenwalt** ➔ Ingmar Heinrich • 3 months ago

It also ignores the fact that the market can scale. If you are on 50% of mainframes, you can expand the market by 10x and double several more times with the invention of a PC. There are lots of ignorant business cases made that ignore this and cite Norvig's Law. "You'll never make your money because even if you had 100% of the market it wouldn't be profitable."



wouldn't be profitable.

If Amazon had 100% of the 1995 online book market it wouldn't be worth \$1Trillion. Even if it replaced WalMart's in-store sales it wouldn't be worth \$1Trillion. It's worth \$1Trillion dollars because it controls AWS... the cloud computing market didn't even exist when Amazon was founded.

^ | v • Reply • Share ›

[Show more replies](#)



zn • 3 months ago

## ABOUT ME



(/author/tim)

**Tim Sommer**

Belgium, Antwerp

I'm a Web Developer and Windows Insider MVP with +8 years of experience in the .NET framework, with a strong focus on HTML5 and JavaScript development.

**f** (<https://www.facebook.com/s0mmerTim>) **t** (<https://twitter.com/sommertim>)

**🔗** (<https://blog.timsommer.be/rss>)

## TOP POSTS





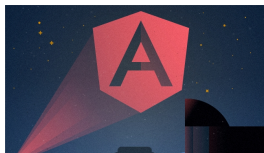
## Famous laws of Software development (/famous-laws-of-software-development/)

Dec 18, 2017



## Using \$compile to compile HTML strings in Angular (/using-compile-to-compile-html-strings-in-angular/)

Oct 30, 2014



## Using ngSanitize to render HTML strings in Angular (/using-ngsanitize-to-render-html-strings-in-angular/)

May 11, 2014

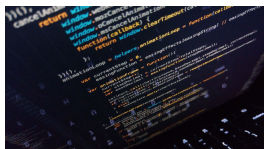
### RECENT POSTS



## (Don't?) Keep It Simple Stupid! (/dont-keep-it-simple-stupid/)

May 07, 2019

(/dont-keep-it-



## Declarative JavaScript Functions (/declarative-javascript-functions/)

Feb 20, 2019

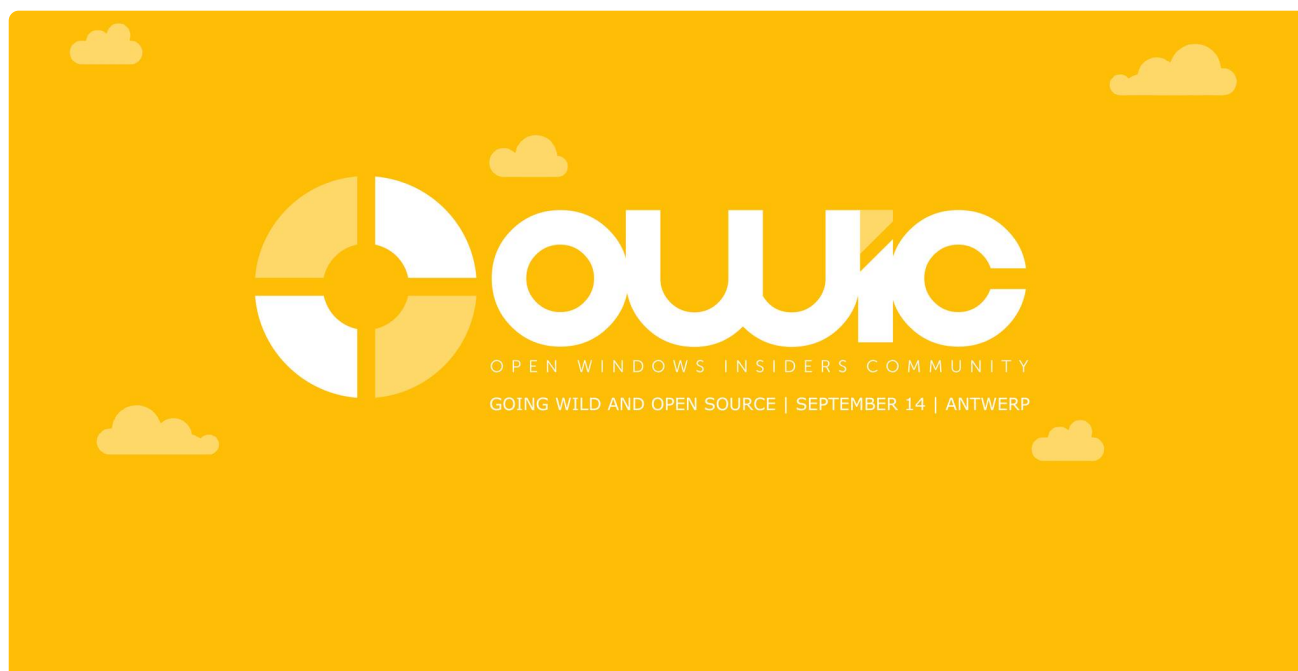


## 2018 - In Search for balance (/2018-in-search-for-balance/)

Jan 25, 2019

### BADGES





(<http://www.owic.be>)



(<https://internetdefenseleague.org>)

## TAG CLOUD



## RECENT TWEETS



## Tweets by @sommertim

Tim Sommer Retweeted

**Uncle Bob Martin**

@unclebobmartin

Never ask permission to refactor. Never ask permission to write tests. You do these things because you KNOW they are the best way to go fast.

When you ask permission, you are asking someone else to take responsibility for your actions.

Jun 1, 2019

Tim Sommer Retweeted

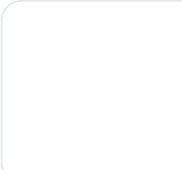
**John O'Nolan**

@JohnONolan

Replying to @sbourke and 2 others

There's a handful of featured sites here:

[ghost.org/customers/](https://ghost.org/customers/) - you can find thousands more on [builtwith.com](https://builtwith.com) :)

	<b>BuiltWith</b> Find out what websites are BuiltWith <a href="https://builtwith.com">builtwith.com</a>
--	---

May 30, 2019

Tim Sommer Retweeted

**Ghost**

@Ghost

"I am one of the most-followed authors on Medium, with 158,000 followers. And yet Medium barely shows my articles to anyone anymore." [@FreeCodeCamp](#) had over 5,000 articles published on Medium — here's why they just moved everything to [@Ghost](#): [freecodecamp.org/forum/t/we-jus...](https://freecodecamp.org/forum/t/we-jus...)

	<b>We just moved off of Medium ...</b> In short: We now have our own open source publishing platform <a href="https://freecodecamp.org">freecodecamp.org</a>
---	--

May 30, 2019

Inspired Web Development

(<https://www.timsommer.be>)

Tim Sommer © 2018.

Proudly published with **Ghost** (<https://ghost.org>)



BACK TO TOP