

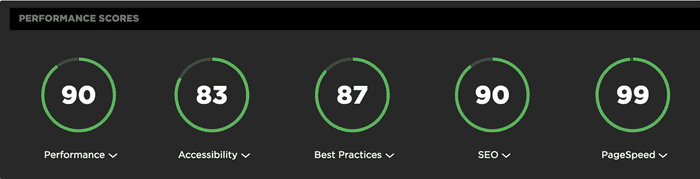
**Speaking Cosmic (JS)**

January 25, 2019

Looking at this blog (made with [Gatbsy.js](https://www.gatsbyjs.org)) and towards Javascript “New Era”, I have been very impressed by how far this language, its frameworks and community had evolved to cover all requirements historically made with PHP on server side. This blog gives you a taste of what can be achieved today with those technologies, and its purpose is also to give you the key in order to build your own tools set. Let’s have a look at cons, benefits, and new philosophies emerging from this interesting era.

**Performances first**

As a SEO for example, you might had faced difficulties trying to improve website performances through technical achievements. The main reason is, most of the time, performances have not been considered at the beginning of your development process, during the foundations of your project. In the same way that [code-first approach](https://blog.devart.com/entity-framework-code-first-support-for-oracle-mysql-postgresql-and-sqlite.html), performances-first approach will clean the path to better SEO website performances, keeping architecture clean and sticking to project first intentions.



Google Chrome Website Audit

Performances described here from Google Chrome Audit shows the potential of such approach. It stands as the best performances I have seen so far without the efforts. Each element is saved and pushed through actual web development best practices, which guarantee optimal balance during deployment.

**Key Performance Indicators (KPI’s)**

According to today’s web requirements, essential points are being covered natively with Gatbsy framework, numbers gathered by the official team refers to :

* 30% faster load
* 10% longer average session
* 40% lower boucing rate
* 38% more conversions

All of this because their creators think about performances first, efficient routing and deep SEO integrations in every website component. Static publishing for modern web apps, 3rds party CMS integrations and as I said before, performances baked-in (images, code-splitting, etc.).

**Optimization**

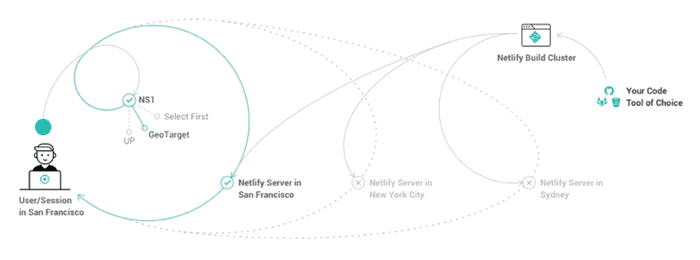
Because performance is key, you might want to tweak your project with those frameworks in order to achieve better user experience which, in a digital era, is the key to reach more visitors and to make them stay, also making them buying more of your products. Here is the prioritized list you must pay attention to :

* fast initial page load time
* fast subsequent page interaction (pre-fetching)
* image optimization
* route-based code splitting

As pre-fetching, Server Side Rendering (SSR) is also a dramatic issue being tackled.

**User Location in Mind**

Make your content accessible from anywhere in the world with publishers which will allows CDN [Content Delivery Network](https://www.netlify.com/blog/2016/04/15/make-your-site-faster-with-netlifys-intelligent-cdn/), making your content available faster from the user location.



Netlify Content Delivery Network

**Share This Post**

# 

# Getting S.O.L.I.D

December 27, 2018

In Object-oriented programming, SOLID is a mnemonic acronym for 5 design principles intended to make software designs more understandable, flexible and maintainable.

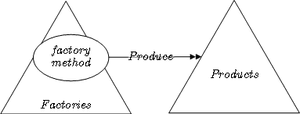
## Design Principles

* S—Single Responsibility
* O—Open/Closed
* L—Liskov Substitution
* I—Interface Segregation
* D—Dependency Inversion

### Single Responsibility

“A class should have one and only one reason to change.”

Meaning that a class should only have one job. The [Factory Pattern](https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)) determine an interesting way to handle repetitive task pattern. Objects factory **parameters definition** quick-start production with minimal constraints.



### Open / Closed

Objects or entities should be open for extension but closed for modification. Open for extension means that we should be able to add new features or components to the application without breaking existing code.

“Closed for modification means that we should not introduce breaking changes to existing functionality, because that would force you to refactor a lot of existing code.” [Eric Elliott](https://medium.com/@_ericelliott)

In simpler words, means that a class or factory function in our case, it should be easily extendable without modifying the class or function itself.

Coding to an interface is an integral part of S.O.L.I.D. Every shape must implements interfaces. You can also use abstract classes, and refer to [RosettaCode](https://rosettacode.org/wiki/Function_composition) to identify the shape corresponding to the langage your are using **function composition** i.e. : **one argument**, only.

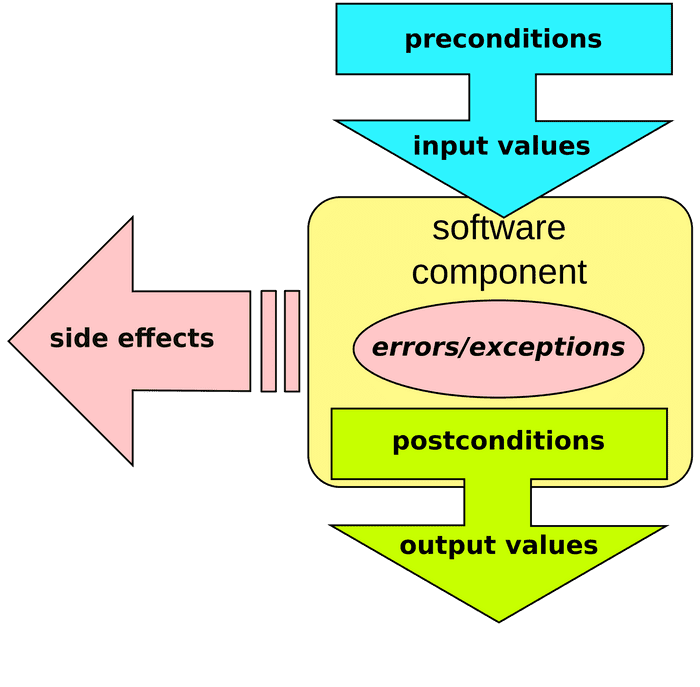


### Liskov Substitution

Derived classes must be substitutable for their base classes. What is wanted here is something like the following substitution property:

“If, for each object A of type S there is an object B of type T, such that for all programs P defined in terms of T, the behavior of P is unchanged when A is substituted for B then S is a subtype of T.” [Barbara Liskov](http://www.pmg.csail.mit.edu/~liskov/)

Each method should have preconditions and posconditions defined. Preconditions must hold true in order for a method to execute, and postconditions must hold true after the execution of a method. It’s what [Robert C. Martin](https://twitter.com/unclebobmartin?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor) suggests as : [Design by Contract](https://en.wikipedia.org/wiki/Design_by_contract).



“When redefining a routine (in a derivative for example), you may only replace its precondition by a weaker one, and its postcondition by a stronger one.” [Bertrand Meyer](https://bertrandmeyer.com/)

Robert C. Martin also suggests that it is helpful to document (comments) the preconditions and postconditions for each method. From my point of view, **YAGNI** principle is right in that context too: ”[You aren’t gonna need it](https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it)”. Forget about comments, it’s systems most common weakness. If your code isn’t strong enough to be read without comments, REWRITE !! before saving anything else.

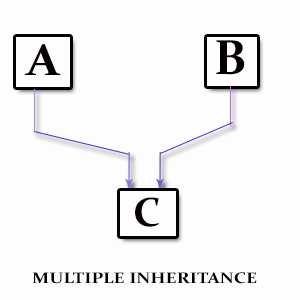
### Interface Segregation

The Interface Segregation Principle was defined by Robert C. Martin while consulting for Xerox to help them build the software for their new printer systems. He defined it as:

“Clients should not be forced to depend upon interfaces that they do not use.”

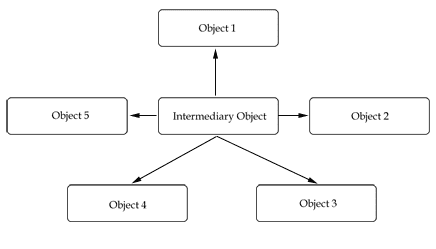
The goal of this is to reduce the side (ripple) effects and frequency of required changes by splitting the software into multiple and independent parts.

For example, let’s say we had an interface called Animal, which would have eat, sleep and walk methods. This would mean that we have a monolithic interface called Animal, which would not be the perfect abstraction, because some animals can fly. Breaking this monolithic interface into smaller interfaced based by role, we would get CanEat, CanSleep and CanWalk interfaces. This would then make it possible for a species to eat, sleep and for example fly. A species would be a combination of roles, instead of being characterized as an animal, which would not necessarily be the best description. At a larger scale, microservices are a very similar case, they are pieces of a system separated by responsibilities, instead of being a great monolith.



By breaking down interfaces, we favor Composition instead of Inheritance, and Decoupling over Coupling. [Coupling](https://gamedevelopment.tutsplus.com/tutorials/quick-tip-the-oop-principle-of-coupling--gamedev-1935) is the principle of reducing how objects directly affect the states and behaviors of other objects. Coupling helps to create code that is easier to read as well as easier to change. It is the principle behind : **Separation of Concerns**.

“One object doesn’t directly change or modify the state or behavior pf another object.”



public class Game {

if (Pac-Man.eats(Power\_Pellet)) {

Ghosts.changeState();

}

}

### Dependency Inversion

Depend on abstractions, not on concretions.

“Abstractions should not depend upon details. Details should depend upon abstractions.”

High level modules should not depend upon low level modules. Both should depend upon abstractions. Let’s say we have a system that handles authentication through external services such as Google, GitHub, etc. We would have a class for each service:

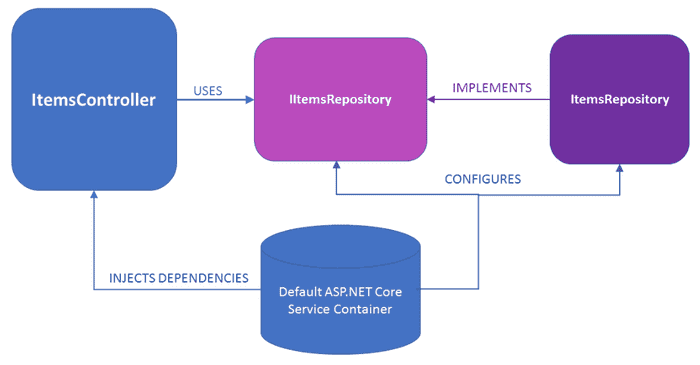
* GoogleAuthenticationService
* GitHubAuthenticationService
* etc.

Now, let’s say that some place in our system, we need to authenticate our user. We have two possibilities:

1. Adapt each service to the authentication process
2. Define an abstraction of the authentication services.

The first possibility is a dirty solution that will potentially introduce technical debt in the future; in case a new authentication service is to be integrated to the system, we will need to change the code, which as a result violates the OCP.

The second possibility is much cleaner, it allows for future addition of services, and changes can be done to each service without changing the integration logic. By defining a AuthenticationService interface and implementing it in each service, we would then be able to use [Dependency Injection](https://philippe.developpez.com/articles/dotnet/injectiondedependances/) in our authentication logic and have our authentication method signature look something like this: authenticate(AuthenticationService authenticationService).



Then, we could authenticate by a specific service like this: authenticate(new GoogleAuthenticationService). This helps us generalize the authentication logic without having to integrate each service separately. By depending on higher-level abstractions, we can easily change one instance with another instance in order to change the behavior. Dependency Inversion increases the reusability and flexibility of our code.

Further reading -> [Don’t blame the dependency injection framework](https://www.continuousimprover.com/2018/05/dont-blame-dependency-injection.html)

## Benefits

Following principles always has benefits. It is no different in software engineering. Following the SOLID Principles gives us many benefits, they make our system :

* reusable
* maintainable
* scalable
* testable
* …

For more about the benefits of each of these principles, make sure to read Uncle Bob’s articles, and communicate about your own experiences.

Thank you Stranger, see you next time, for another coffee-programming session.

Going further: ***DRY Principle, Encapsulation, Premature Optimization, Refactoring, Software Craftmanship, Clean Code and Style Guides***.

## Share This Post

# 

# Road-Map

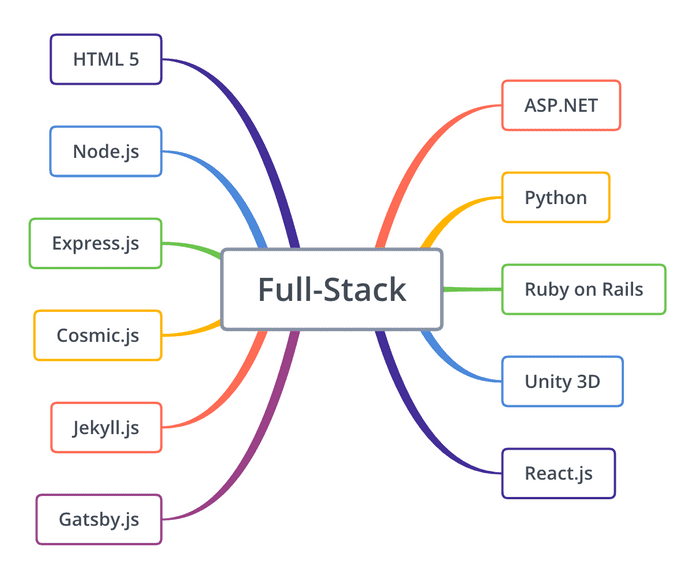
December 08, 2018

This page aims at showing at a glance my research path, founds, and gems. There is a lot of burning topics nowadays ! Main frameworks I am using today are presented in the section below, you can also have a look at my Bookmarks if you want, I will do my best to keep this list updated regularly.

## IDE

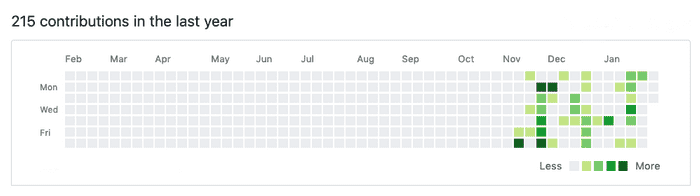
My development environment is based on shared projects between MacOS and Windows, Back/Front-End development and 3D environments. Configuring Windows to execute Linux command, and merging frames in Visual Studio Code makes the bridge between those two. Thank you Microsoft for your last actions towards community developers. Open-source softwares will win, at the very end.

## Frameworks



I currently enjoy front development libraries based on Javascript like [Jekyll](https://jekyllrb.com/) and [Gatsby](https://www.gatsbyjs.org/). Deployment from [Npm](https://www.npmjs.com/) is like having a terminal copilot which does the main part of the job for you. It takes a few time to obtain a stable environment but it really worth the time !

## Overview



This end of the year has been an exciting field of research around web technologies. My personal main target looking at my contributions is to reach the most linear production as possible, that is to say to enhance predictibility and velocity over incoming year and to improve techniques acquisitions through code writing of course, but also through professional meetings and events where developers and managers shares tips and good practices in order to build faster and better softwares.

## Share This Post

# 

# Welcome, Stranger!

November 30, 2018

Welcome to my blog, I hope you will enjoy your stay! Decrease ambient lights, fasten your seatbelts, and follow me inside front and back-end various programming.

## Who am I ?

French generalist engineer, dedicated to software programming since 2012. I started professional coding in a research laboratory environement, where aeronautics or military customers where focus on system ergonomics analysis, and developing innovative features through off-the-shelves methods or absolutely new devices. Among others, I also worked in more creative environment like dance performances, and kids workshop around virtual and augmented reality too.

Learn. Observe and Do Things with Intention.

Comments sections will be added soon. Meanwhile, you can contact me at this adress : [jconan@outlook.fr](mailto:jconan@outlook.fr)(jconan@outlook.fr)



## Share This Post