

Dependencies Demystified

Jason Bock
Developer Advocate
Rocket Mortgage

Personal Info

- <http://www.jasonbock.net>
- <https://www.twitter.com/jasonbock>
- <https://www.github.com/jasonbock>
- <https://www.youtube.com/c/JasonBock>
- jason.r.bock@outlook.com

Downloads

<https://github.com/JasonBock/DependenciesDemystified>
<https://github.com/JasonBock/Presentations>

Overview

- Dependencies
- Containers
- Call to Action

Remember...

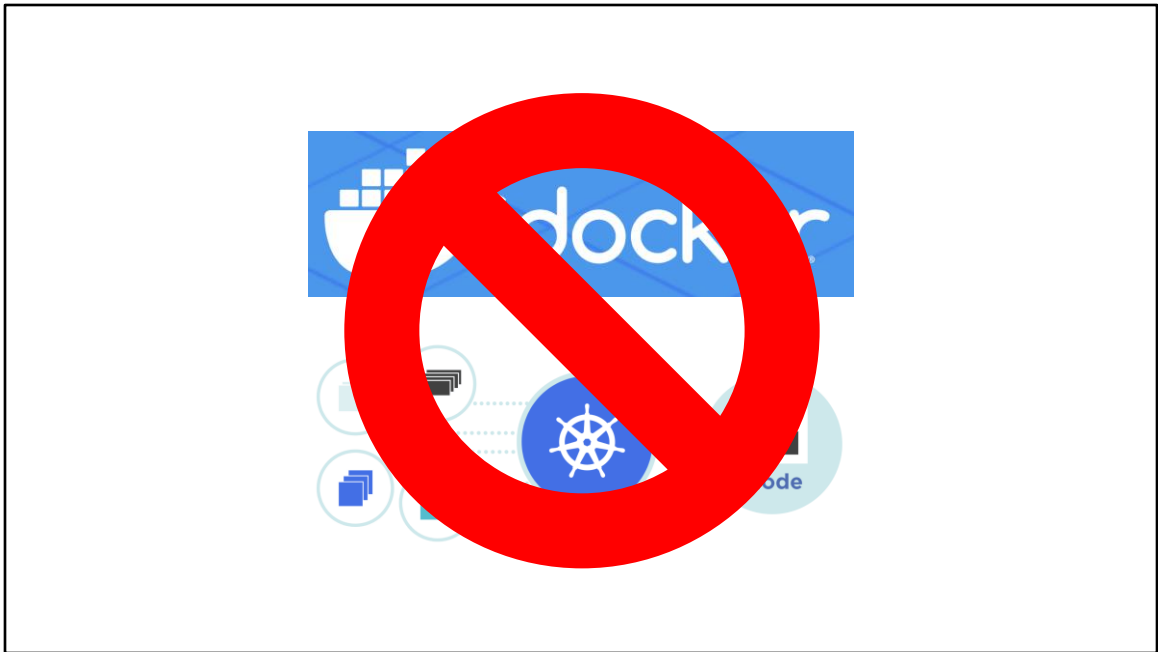
<https://github.com/JasonBock/DependenciesDemystified>

<https://github.com/JasonBock/Presentations>



Just be clear right away, when people hear “containers”, this is what people think of now. That’s not what we’re going to talk about.

<https://www.docker.com/>
<https://kubernetes.io/>



Just be clear right away, when people hear “containers”, this is what people think of now. That’s not what we’re going to talk about.

<https://www.docker.com/>
<https://kubernetes.io/>

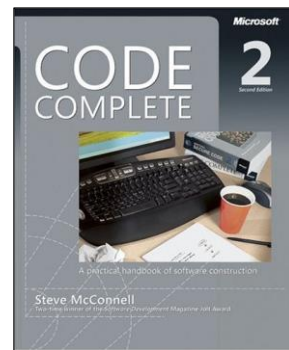
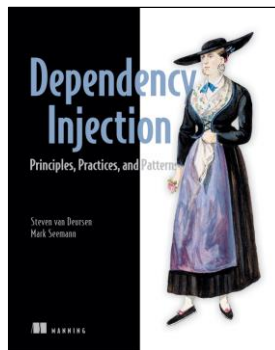
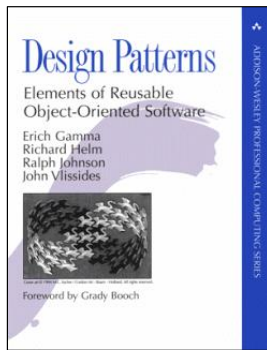
Dependencies



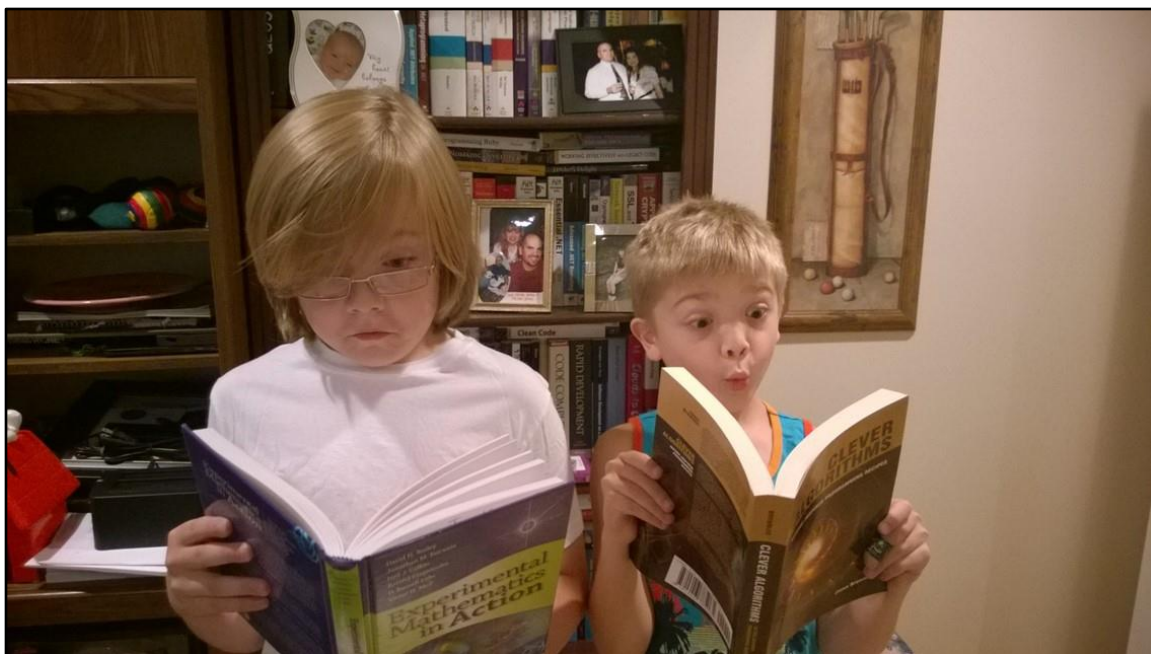
Dependencies....what are they? Why should I care? Don't they just make everything harder? I can easily connect to my service or database in code, what's with all this misdirection, extra layers? Is this another form of job security?

Even though it really is a simple concept, it took me a while to really get my head around dependencies. But if you start to recognize dependencies in code and handle them correctly, the benefits are tremendous. And in 2021, dependencies are everywhere, so at this point, it's something you should be familiar with – at the very least, you've seen them (though you may not have understood how it all works).

<https://unsplash.com/photos/ufgOEZuHgM>



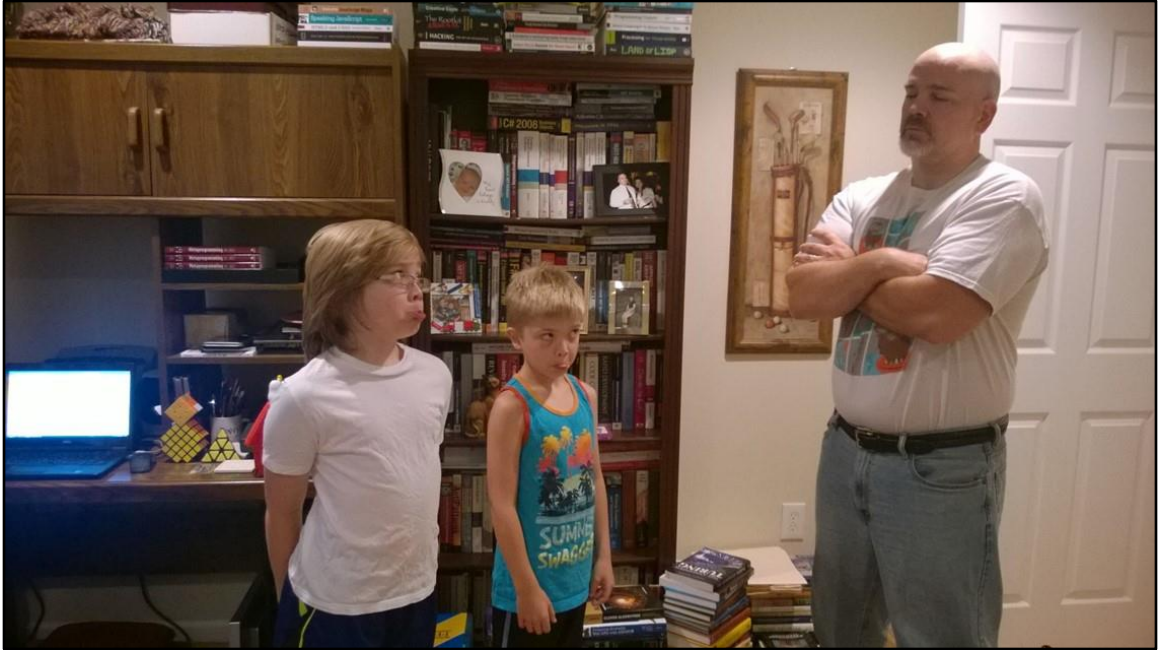
I have references in the notes section of the last slide, but these books are great at talking about dependencies, even if they didn't do it directly. Let's focus in on the dependency aspect with a real-world example.



I have two kids. (These pics were taken in 2015, so they're much older now, but I have to keep using these pictures)



They're dependent on me. Sometimes they make requests from me, like "give me money!"



I may return 0, or RoomNotCleanedException. (or even -2)



Or I may pay them for their hard work (a quarter is a quarter!)

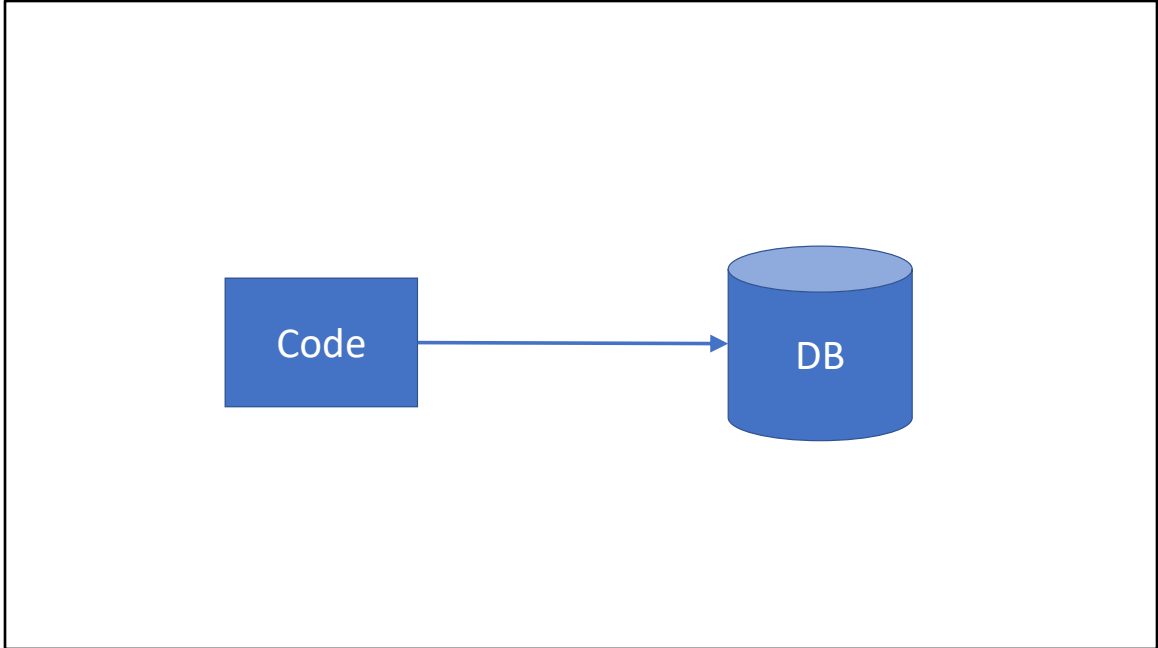


If I don't work, make money, pay the mortgage, get groceries, etc. I turn into a null reference, and that's not good.

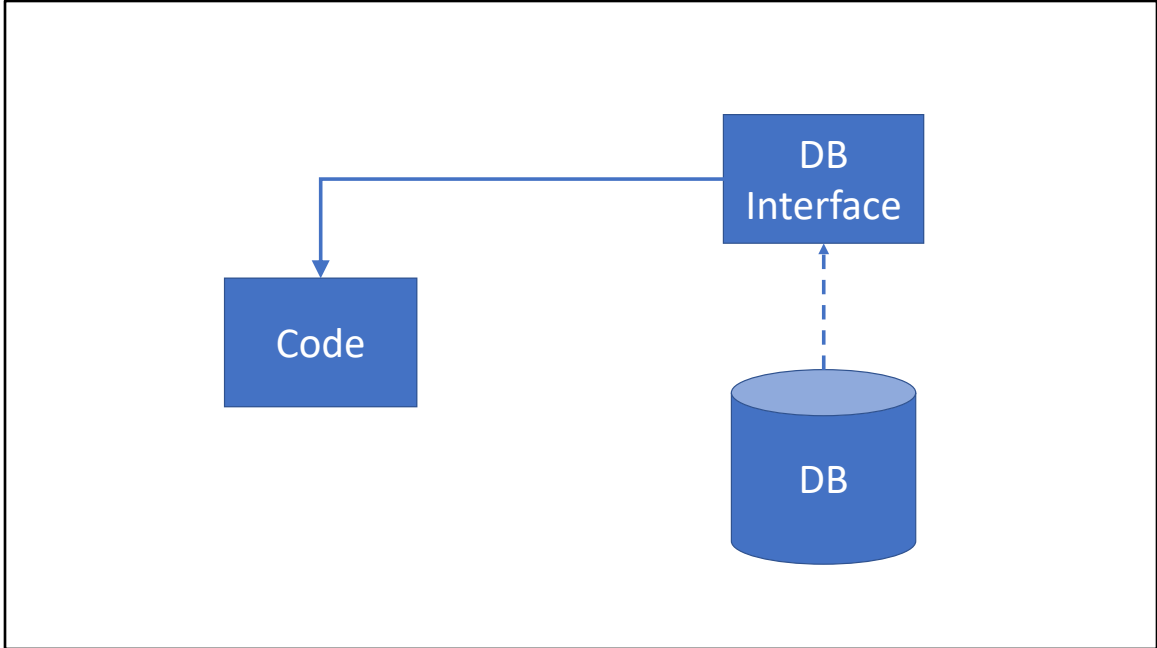
They're independent
resources that
perform services you
need

That's all dependencies are. This is the simplest answer I could come up with off the top of my head.

"Independent" == you don't "own" them

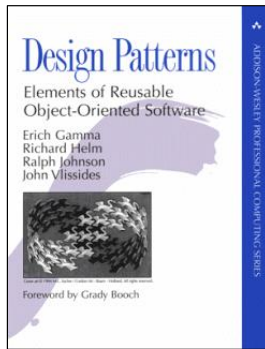


A canonical example is DB access. You can write code that talks to a database directly, or...



You can use an interface instead. The implementation is given to you, but you don't know or care how it's implemented. You just use the interface definition.

Note that this is usually passed into an object on construction. There are other ways to do it, but the majority of the time it's constructor injection.



"Program to an
interface, not an
implementation"

This is one of the best quotes in Design Patterns. The patterns themselves are good, but they're all based on this premise. You don't want to know how the patterns are implemented; you just want to use them in a pluggable way.



And why is this beneficial? One, you can blame others 😊. In a way, this is a good thing. You don't want that responsibility! Put the details of how to create the dependency somewhere else...in a central location...we'll get back to that later.

<https://www.pexels.com/photo/photo-of-man-pointing-his-finger-1259327/>

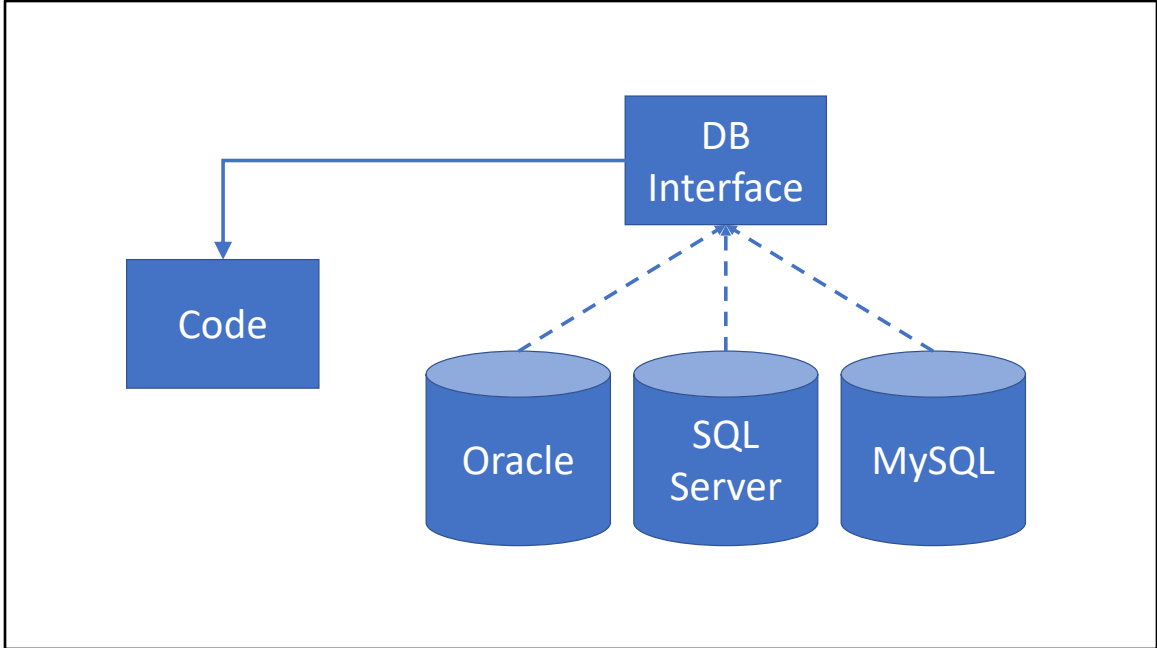
PASS



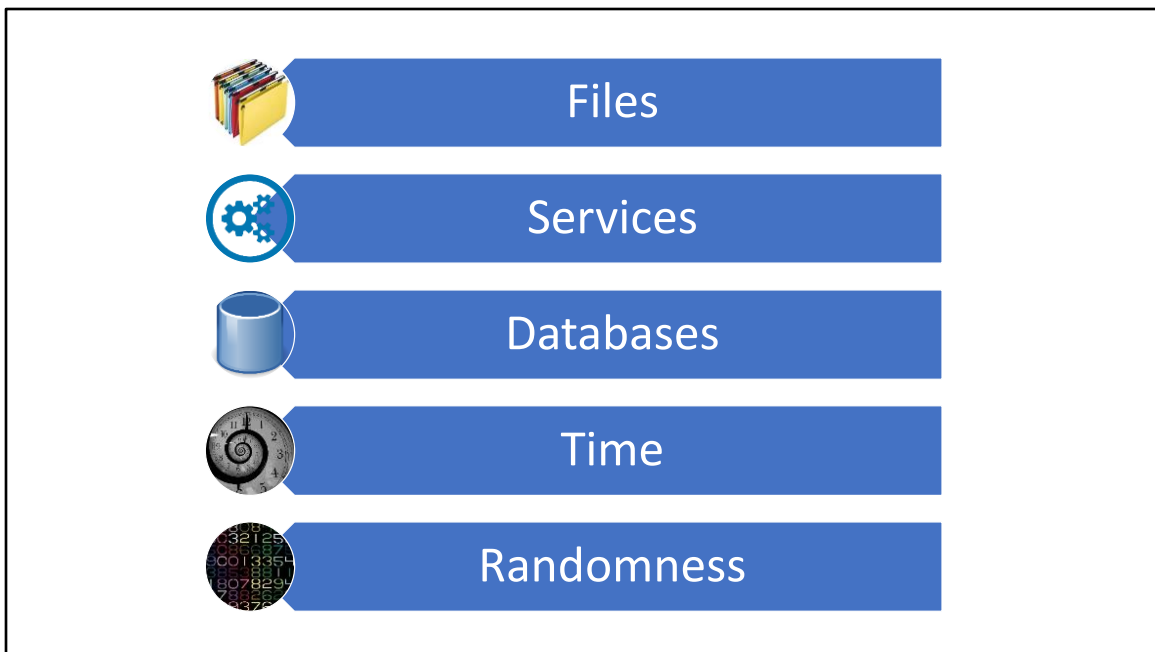
FAIL



It can make testing easier. You don't have to have N number of dependencies up and running just to see if the code under test works. Integration/end-to-end testing is definitely needed, but having a suite of fast unit tests is a beautiful thing to have.



It also has the benefit that you can use different implementations, though I've personally not found that to be a huge win (YMMV)



What are some examples of typical dependencies?

- Files: Code that uses files doesn't need to actually create the file. Interfaces are sufficient for this (like streams, or libraries like `System.IO.Abstractions` - <https://github.com/tathamoddie/System.IO.Abstractions>)
- Services: Talking to a back-end REST or WCF service isn't the goal: getting or updating information via these services is
- Databases: see above 😊
- Time: Time can be hard to manage, especially in testing scenarios, where "`Date.UtcNow`" just doesn't give enough fine grain control. Having a dependency provide time is a good thing (Reactive Extensions provides this via `IScheduler`)
- Randomness: Ever try to test code that uses random data? Or what if you want to switch the random generator out?

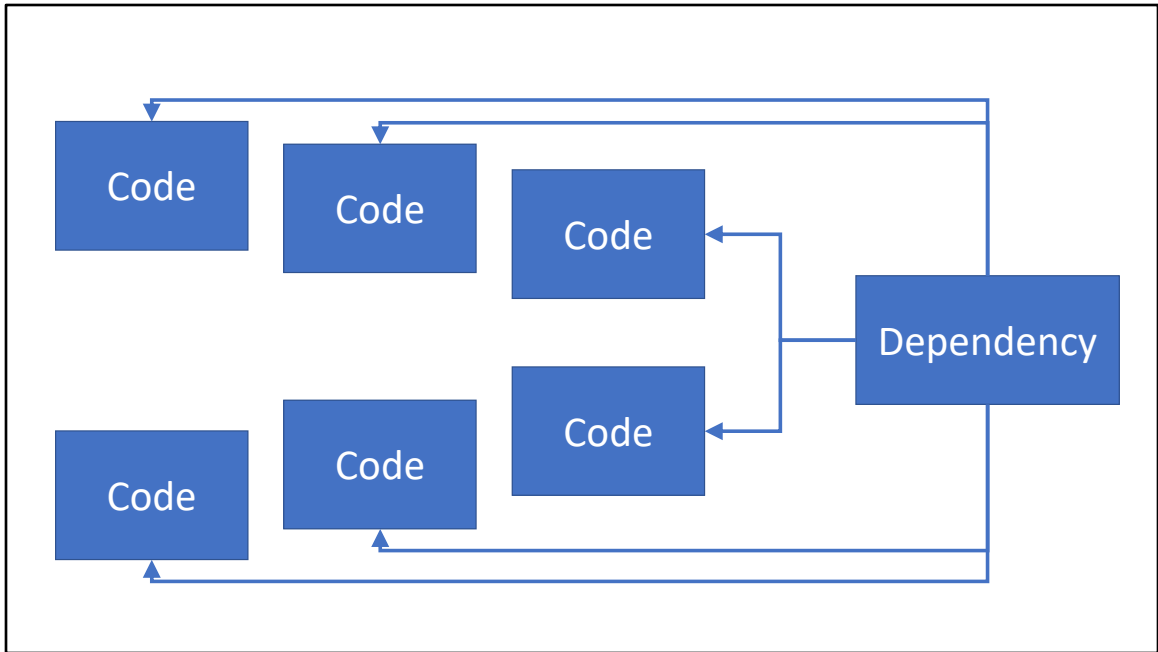
This isn't a complete list. You may choose to define other dependencies

Demo: Refactoring Code to Use Dependencies

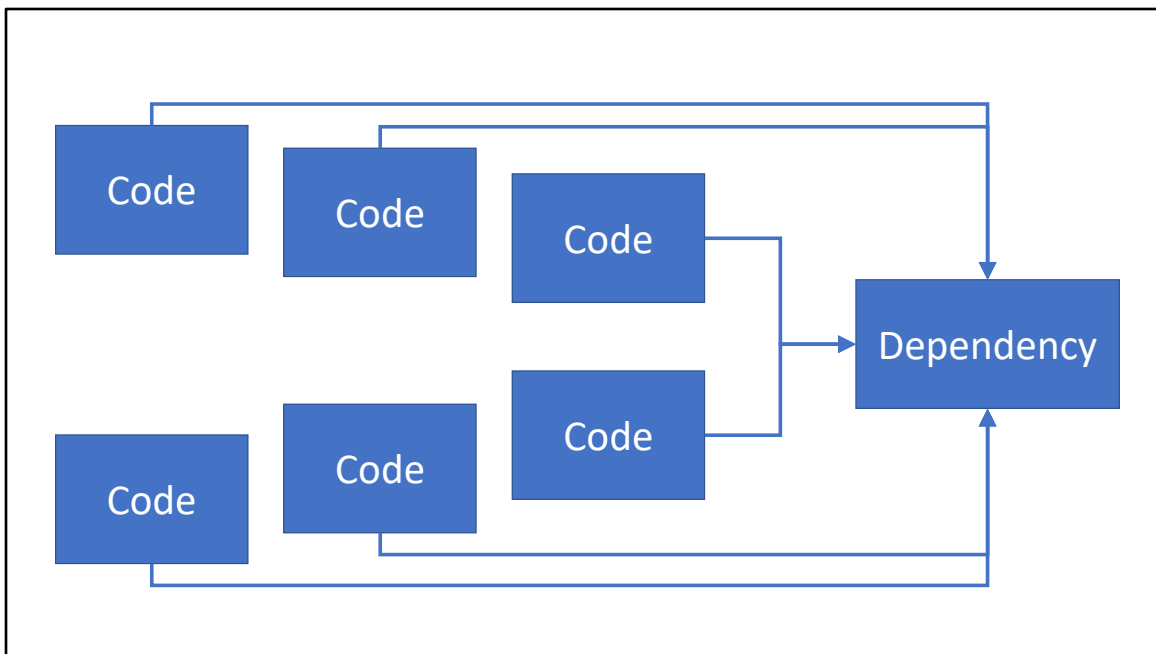
Dependencies Demystified

Let's take code that uses dependencies directly, and then "turn it around".

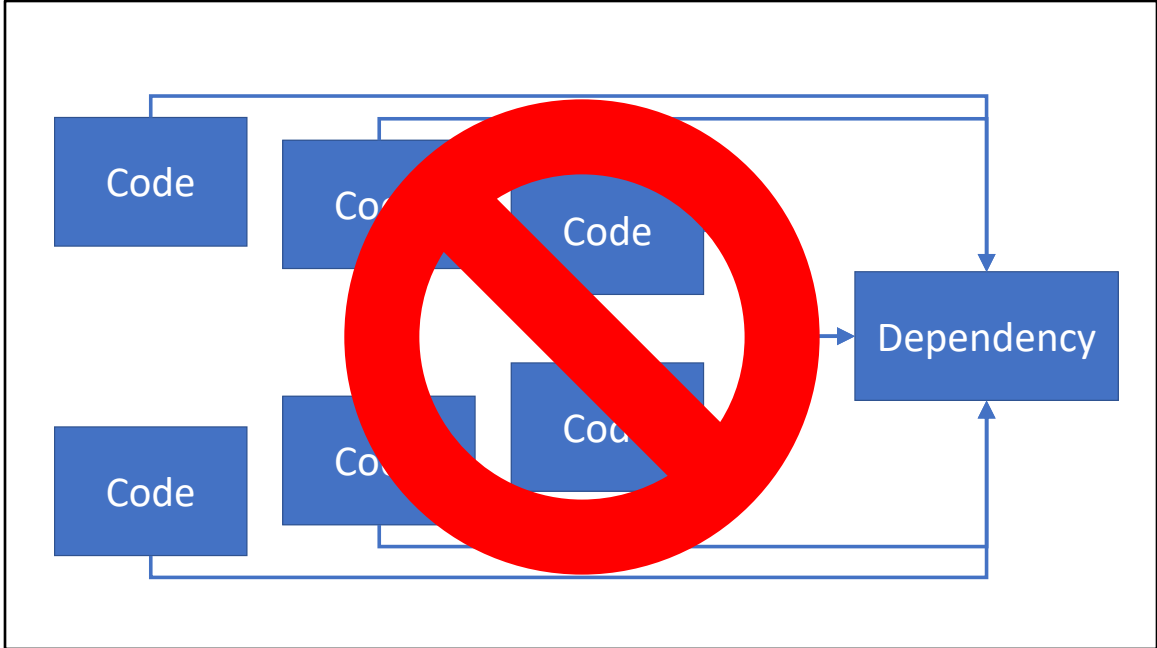
Containers



It's typical that dependencies are needed at multiple locations in code



Another thing devs will do is use a “service locator” pattern. There’s a global container defined that code knows about and resolves dependencies there.



DON'T



Hides dependencies

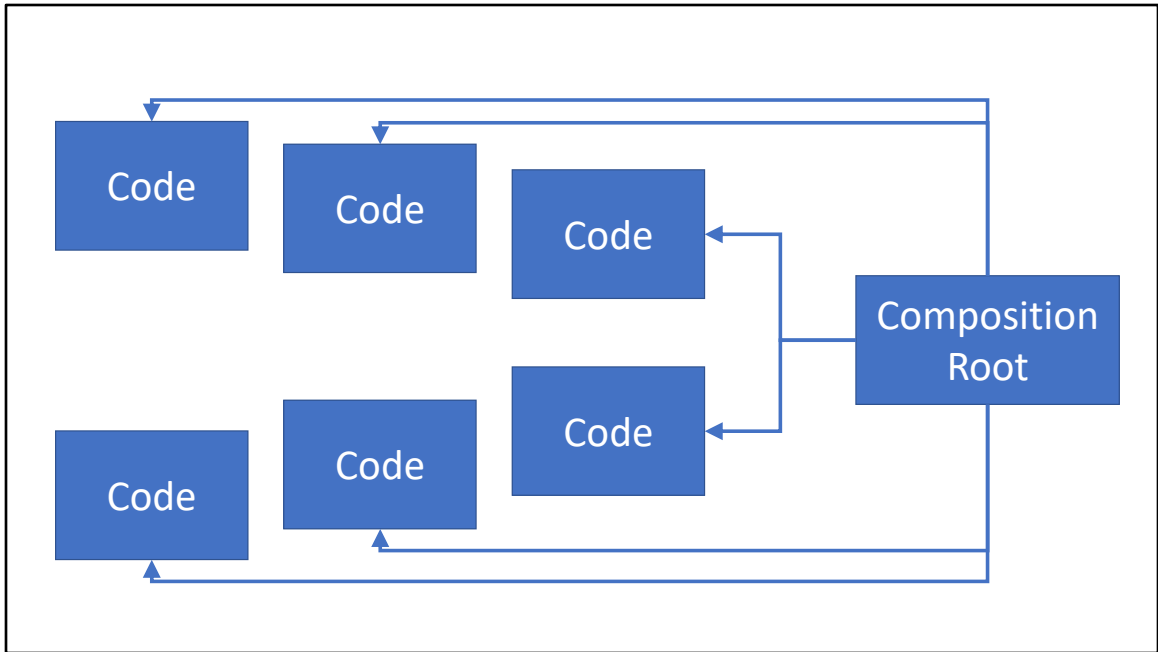
Forced DI knowledge

Unit testing difficulties

The first problem is that the SL approach isn't explicit. You don't know what the class needs to function properly. The DI version is explicit.

You also need to know that the class must have the SL configured.

Finally, if you run tests in parallel, you could end up with race conditions where one tests configures the SL but another overwrites it



So let's use a container that will manage all the defined dependencies in an application. That's the responsibility of an IoC container



Register



Resolve



Release

Containers should have the following three features:

- Register – set up all dependencies
- Resolve – get a dependency out of the container
- Release – manage disposable resources

<https://www.pexels.com/photo/antique-banking-blur-business-210588/>

<https://unsplash.com/photos/veNb0DDegzE>

<https://unsplash.com/photos/4BDc2zKbFq0>

Demo: Using ServiceCollection in .NET

Dependencies Demystified



With containers, DI, etc., you need to be aware of relationship types and what they mean. The ServiceCollection covers most cases, but there are some that other containers handle.

- Transient
- Singleton
- Lazy
- And others

<https://www.pexels.com/photo/close-up-photo-of-two-person-s-holding-hands-1667849/>



Along with this is also the concept of lifetimes, and this is something to be aware of.

<https://unsplash.com/photos/LPRrEJU2GbQ>

```
public sealed class DependentChild
{
    private readonly IParent parent;

    public DependentChild(IParent parent)
    {
        this.parent = parent ??
            throw new ArgumentNullException(nameof(parent));
    }

    // ...
}
```

Singleton
Parent



Random
Parent

Dependents should be as ignorant as possible about the dependency they're given. They shouldn't care if the parent would always be the same one for the lifetime of the app, or if

```
public sealed class DependentChild
{
    private readonly Lazy<IParent> parent;

    public DependentChild(Lazy<IParent> parent)
    {
        this.parent = parent ??
            throw new ArgumentNullException(nameof(parent));
    }

    // ...
}
```

Registered
Parent




Dependents should also be able to control the usage of a dependency – e.g. if the dependency isn't always used, it should be able to specify that it's a Lazy<>, even if the dependency wasn't registered that way.

```
public sealed class DependentChild
{
    private readonly IParent parent;

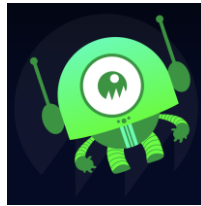
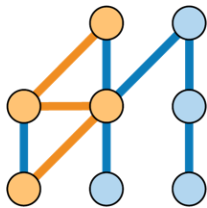
    public DependentChild(IParent parent)
    {
        this.parent = parent ??
            throw new ArgumentNullException(nameof(parent));
    }

    // ...
}
```

IDisposableParent :
IParent, IDisposable



Dependents should not care if the dependency is disposable. In fact, it can't, as this example shows. But SOMEBODY has to care that it's disposable.



StructureMap, Autofac, Unity, and a whole bunch more. There are a lot of custom containers with different features.

I should emphasize that you do NOT have to use one of these. ServiceCollection provides a set of features that can handle a number of features, directly or indirectly.

<https://structuremap.github.io/documentation/>

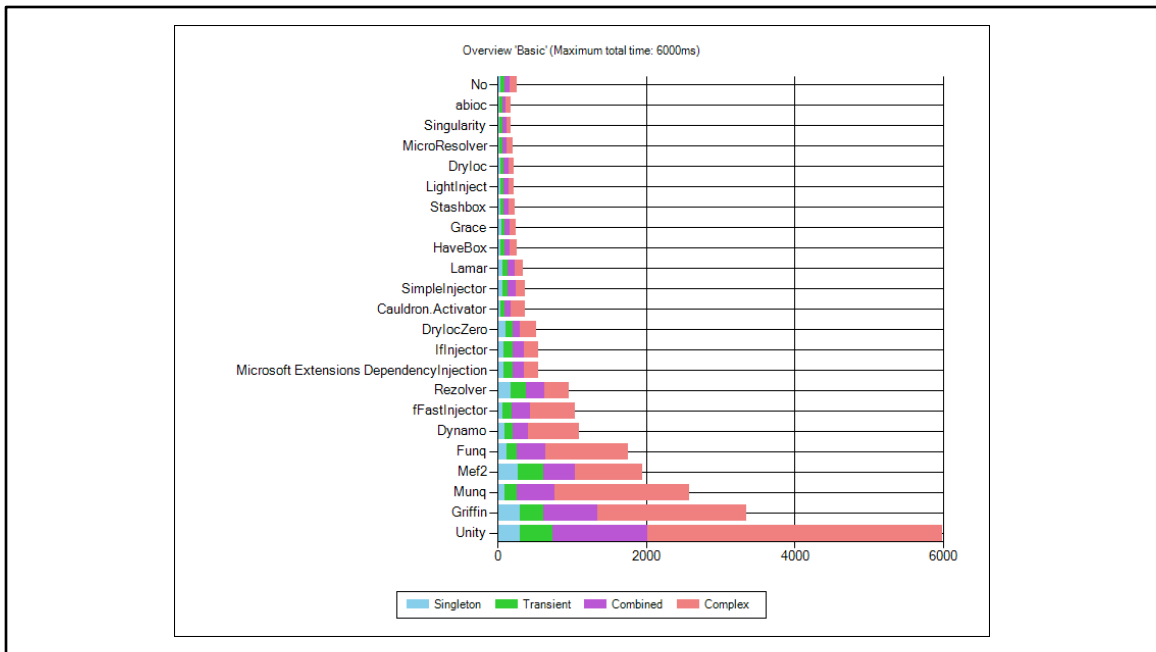
<https://autofac.org/>

<https://github.com/unitycontainer/unity>



Of course, which one you pick is really up to you. There's no good reason to spend a significant amount of time – the most popular ones essentially do the same thing in their own ways. Fighting over a specific container isn't worth the time.

<https://www.pexels.com/photo/offended-young-indian-couple-sitting-on-sofa-4308050/>



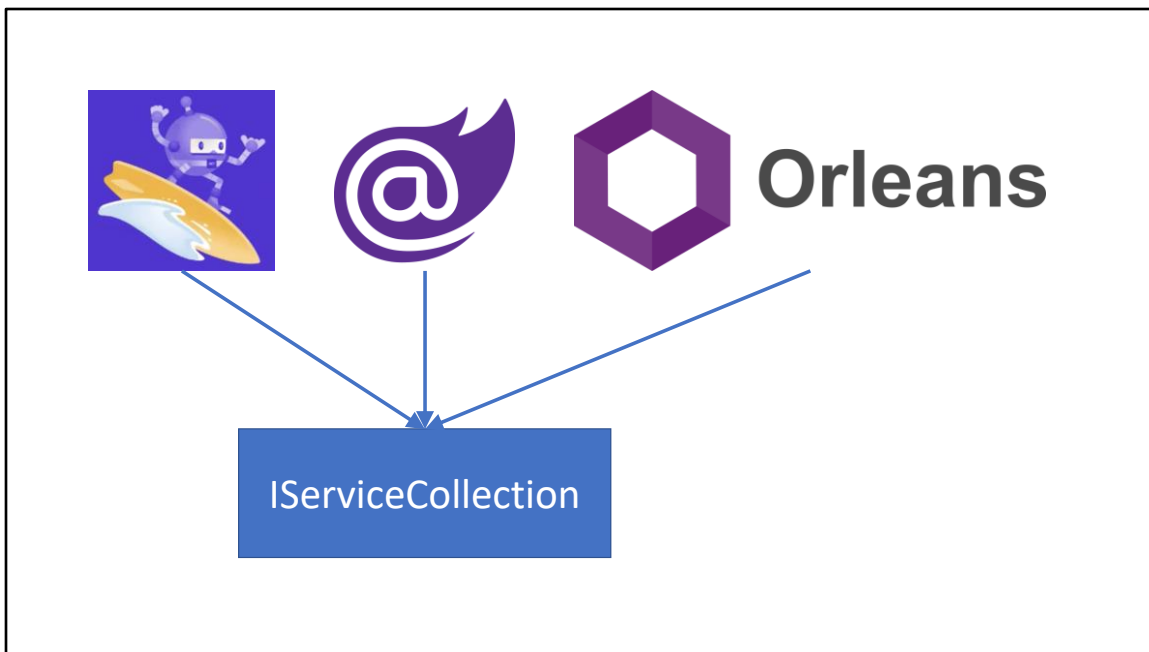
There are great sites that have done a lot of performance investigations into each framework, so check them out and use that to guide your choice.

<https://www.palmmmedia.de/Blog/2011/8/30/ioc-container-benchmark-performance-comparison>

Demo: Using Autofac

Dependencies Demystified

So let's see how we can use Autofac to register dependencies and resolve them using a `Lazy<T>` and a `Func<string, T>`, where the former isn't registered, but the latter is:

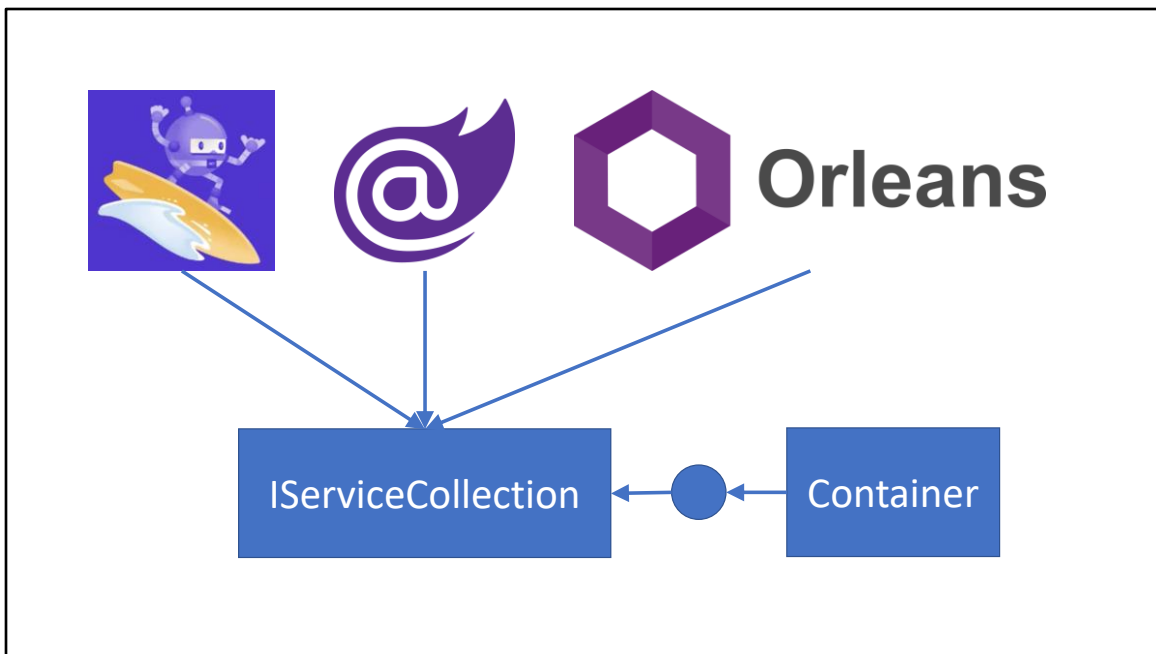


Now, most of us aren't writing console applications for production; other application hosts are used. At this point, all of them integrate with `IServiceCollection`

<https://github.com/dotnet/maui>

<https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor>

<https://dotnet.github.io/orleans/>



Most modern hosts have integration points with NuGet packages that make it pretty simple to put your favorite IoC in play with the application.

Demo: Autofac and IServiceCollection Integration

Dependencies Demystified

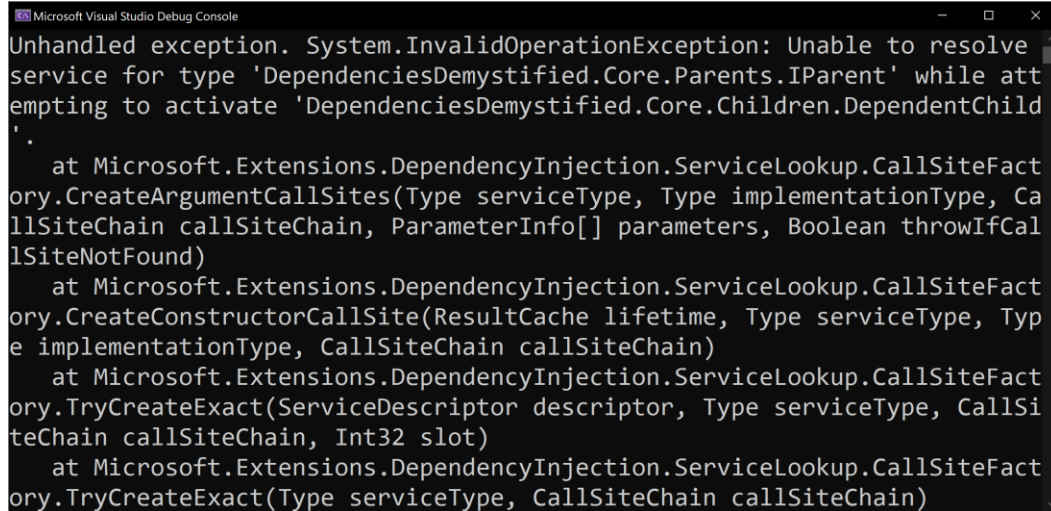
Use Autofac with IServiceCollection

```
var services = new ServiceCollection();
services.AddScoped<IChild, DependentChild>();
var provider = services.BuildServiceProvider();

// Later on...

using var scope = provider.CreateScope();
var child = scope.ServiceProvider.GetService<IChild>();
```

So, all this is good, but....what happens if you forget to register a dependency? In this case, someone forgot to register something for IParent.

A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the text "Microsoft Visual Studio Debug Console" and standard window controls (minimize, maximize, close). The console displays an unhandled exception message in white text on a dark background. The message reads: "Unhandled exception. System.InvalidOperationException: Unable to resolve service for type 'DependenciesDemystified.Core.Parents.IParent' while attempting to activate 'DependenciesDemystified.Core.Children.DependentChild'." Below this, the stack trace is shown with four lines of code, each preceded by "at". The stack trace points to the Microsoft.Extensions.DependencyInjection namespace, specifically to the CallSiteFactory class and its methods: CreateArgumentCallSites, CreateConstructorCallSite, TryCreateExact (twice).

```
Microsoft Visual Studio Debug Console

Unhandled exception. System.InvalidOperationException: Unable to resolve
service for type 'DependenciesDemystified.Core.Parents.IParent' while att
empting to activate 'DependenciesDemystified.Core.Children.DependentChild
'.
    at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFact
ory.CreateArgumentCallSites(Type serviceType, Type implementationType, Ca
llSiteChain callSiteChain, ParameterInfo[] parameters, Boolean throwIfCal
lSiteNotFound)
    at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFact
ory.CreateConstructorCallSite(ResultCache lifetime, Type serviceType, Typ
e implementationType, CallSiteChain callSiteChain)
    at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFact
ory.TryCreateExact(ServiceDescriptor descriptor, Type serviceType, CallSi
teChain callSiteChain, Int32 slot)
    at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFact
ory.TryCreateExact(Type serviceType, CallSiteChain callSiteChain)
```

With `ServiceCollection`, you'll get an `InvalidOperationException`, but this doesn't happen until runtime. Wouldn't it be nice to know this at compile time? (Or when you're typing code?)



Source generators in C# allow you to generate code at compile time, and StrongInject uses that to perform container verification.

<https://github.com/YairHalberstadt/stronginject>

Demo: Using StrongInject

Dependencies Demystified

Use StrongInject

Call To Action



Break your dependencies. You don't want to be chained to them.

https://unsplash.com/photos/6y5iySR_UXc

“The reason that I use IoC is not to encourage testing, it is not to break dependencies, it is not to get separation of concerns...I am using IoC because it makes all of the above so *easy*.”

It's a pattern that makes things so much easier.

https://unsplash.com/photos/4Zaq5xY5M_c

<https://ayende.com/blog/2887/dependency-injection-doesnt-cut-it-anymore>

Dependencies Demystified

Jason Bock
Developer Advocate
Rocket Mortgage

Remember...

- <https://github.com/JasonBock/DependenciesDemystified>
- <https://github.com/JasonBock/Presentations>
- References in the notes on this slide

References

- * [Dependency injection guidelines](<https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection-guidelines>)
- * [Having Fun with Microsoft IoC Container for .NET Core](<https://sahansera.dev/dotnet-core-ioc-container/>)
- * [Should I be Checking Injected Dependencies for Null?](<https://stevetalkscode.co.uk/null-injection>)
- * [C# developers! Your scoped components are more dangerous than you think.](<https://blogs.endjin.com/2020/03/csharp-scoped-di-components-are-dangerous/>)
- * [The Inversion of Control pattern in the test of time](<https://ayende.com/blog/189697-C/the-inversion-of-control-pattern-in-the-test-of-time>)
- * [Runtime analysis and leak detection for Autofac](<https://nblumhardt.com/2019/03/runtime-analysis-for-autofac/>)
- * [Don't blame the dependency injection framework](<https://www.continuousimprover.com/2018/05/dont-blame-dependency-injection.html>)

- * [Dependency injection in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection)
- * [Essential .NET - Dependency Injection with .NET Core](https://msdn.microsoft.com/en-us/magazine/mt707534.aspx)
- * [.Net Core Dependency Injection](https://stackify.com/net-core-dependency-injection/)
- * [IoC Container Benchmark Performance comparison](http://www.palmmedia.de/Blog/2011/8/30/ioc-container-benchmark-performance-comparison)
- * [Inversion of control containers – Best practices](http://www.dotnetcodegeeks.com/2012/05/inversion-of-control-containers-best.html)
- * [Building an IoC container in 15 lines of code](https://ayende.com/blog/2886/building-an-ioc-container-in-15-lines-of-code)
- * [Service Locator is an Anti-Pattern](https://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/)