

Axes

Build software like a bag of marbles,
not a castle of LEGO®

-

Hannes Lowette

Disclaimer #1



AXXES_

Collaboration > coexistence!



AXXES_

Disclaimer #2



AXXES_

PSA

LEGO is a brand name
used as an adjective
there is no plural, 'LEGOs'



Axes

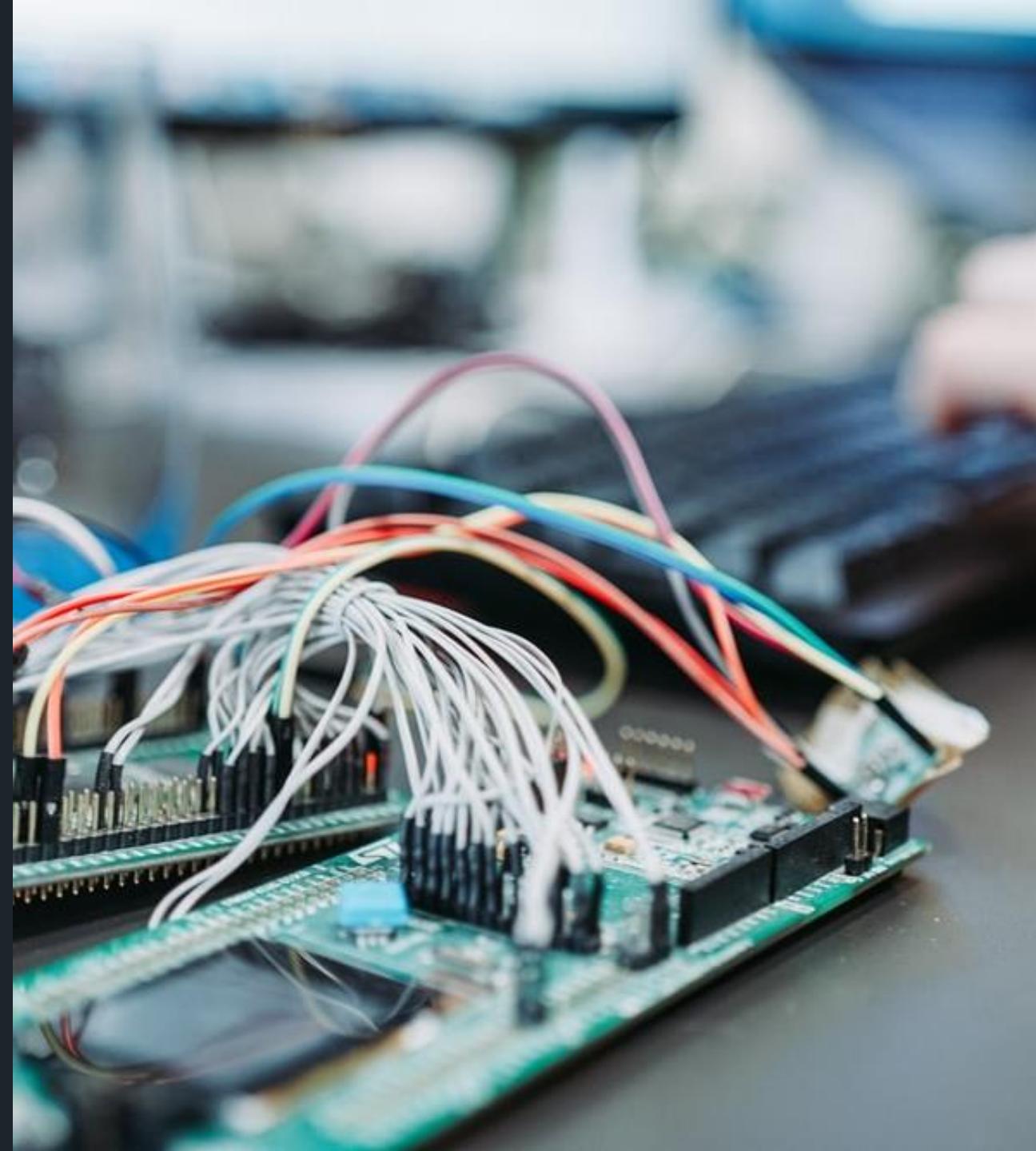
Understanding the problem

—

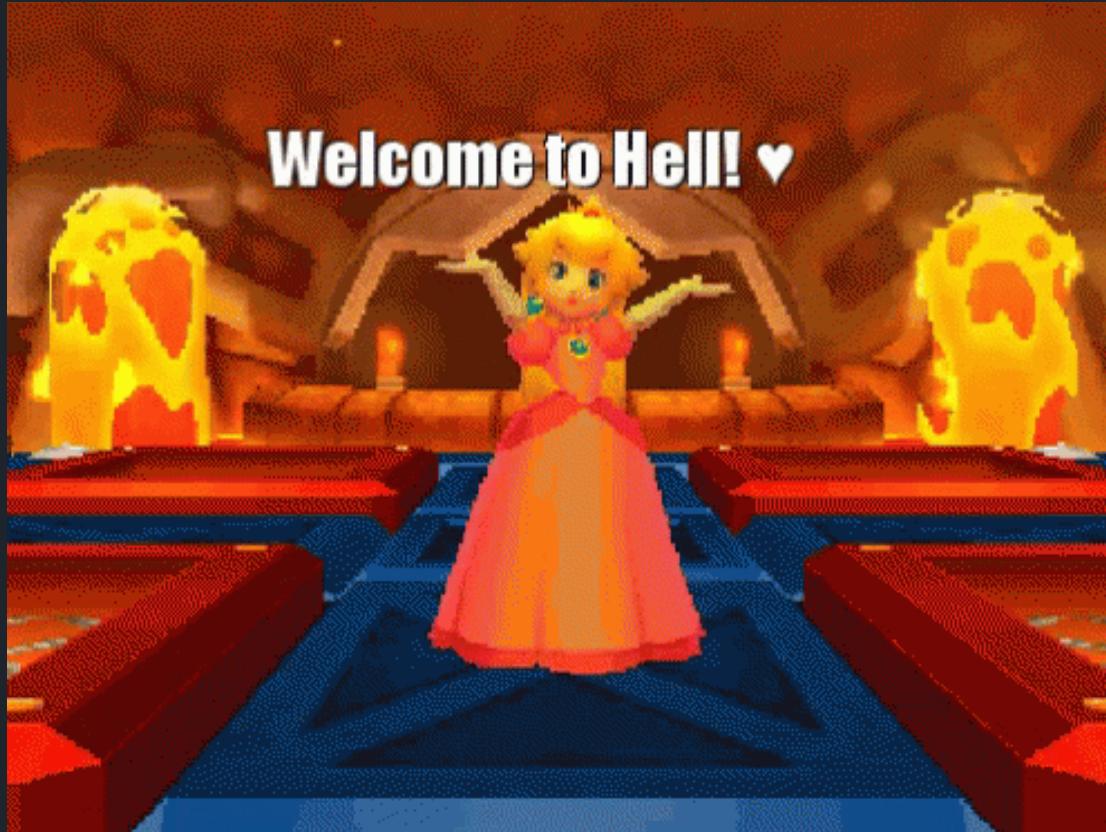
What were we trying to solve?

What were we working on?

- Huge product
- Goal: managing IoT devices
- A lot of **implementations**
(= specific types of devices)
- 1 multi-tenant deployment



How did we roll?



- New **device types** all the time
- Most projects didn't go live
- Code didn't get removed because:
 - Tightly coupled
 - Reused by other devices
 - Money had been invested

→ **Maintenance hell!**

What frustrated me about this?



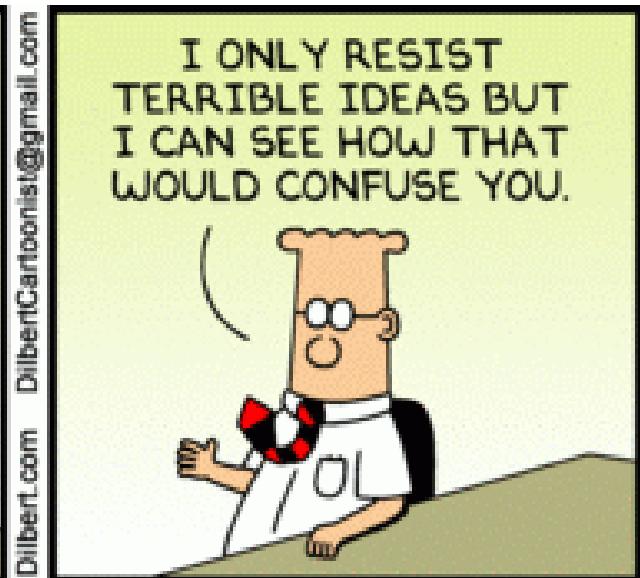
One of my most productive days was
throwing away 1,000 lines of code.

Ken Thompson



Product management

- Not ready to change
- New devices would keep coming
- They expected us to keep the code



AXXES

What the dev team wanted

- Implement new devices quickly
- Be able to remove them easily
- Limit dependencies between devices
- Clean abstractions



→ Stop Pollution!

AXXES

So, microservices, right?



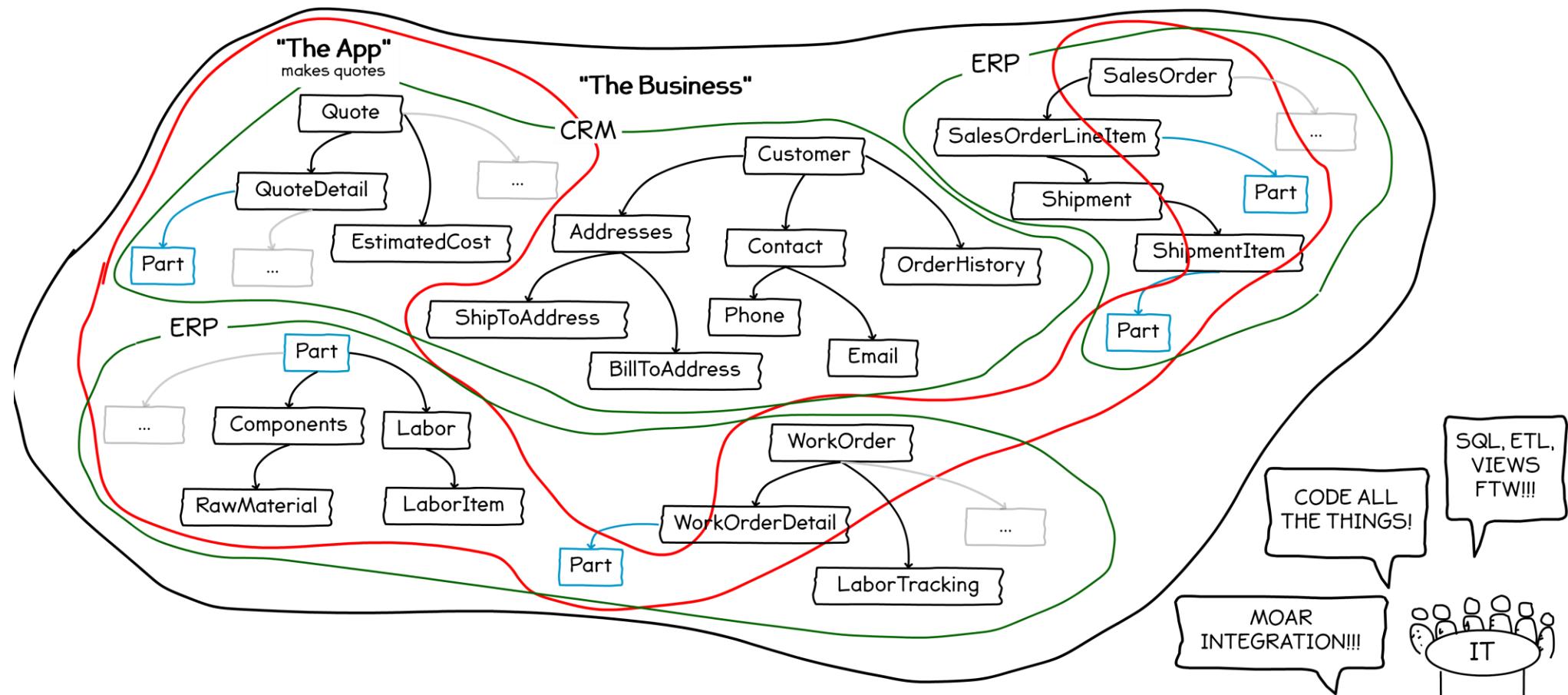
Axes

The OO path to success

—

Steps in our evolutions as developers

Step 1: Demoware



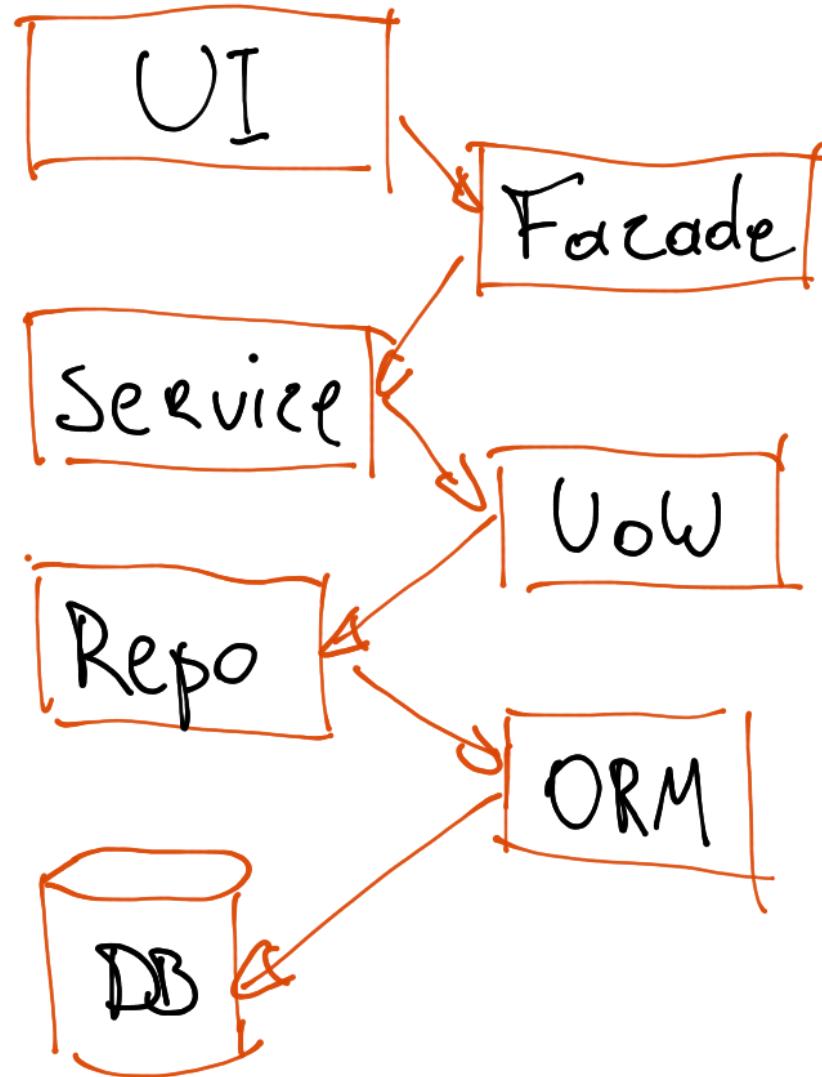
Step 2: Layers



AXXES_

Step 3: "SOLID"

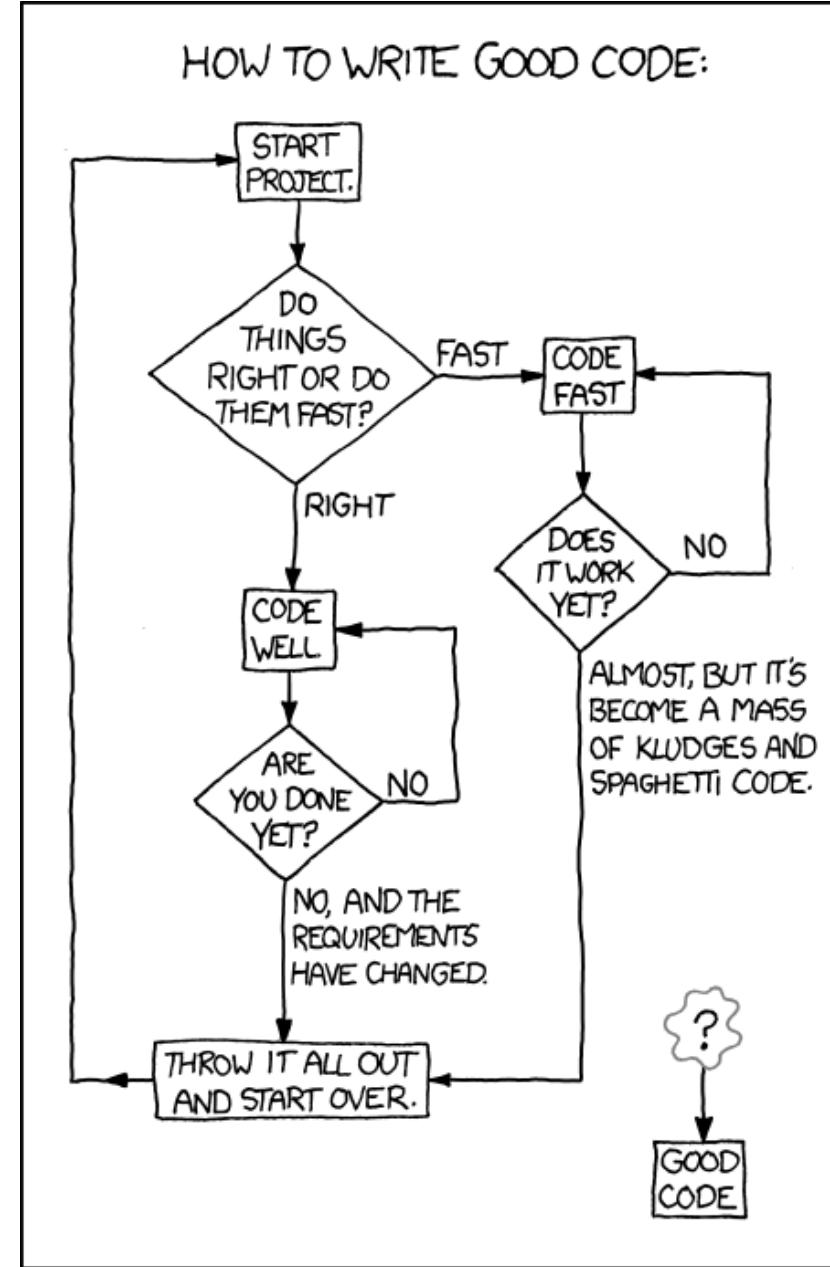
ABSTRACTIONS



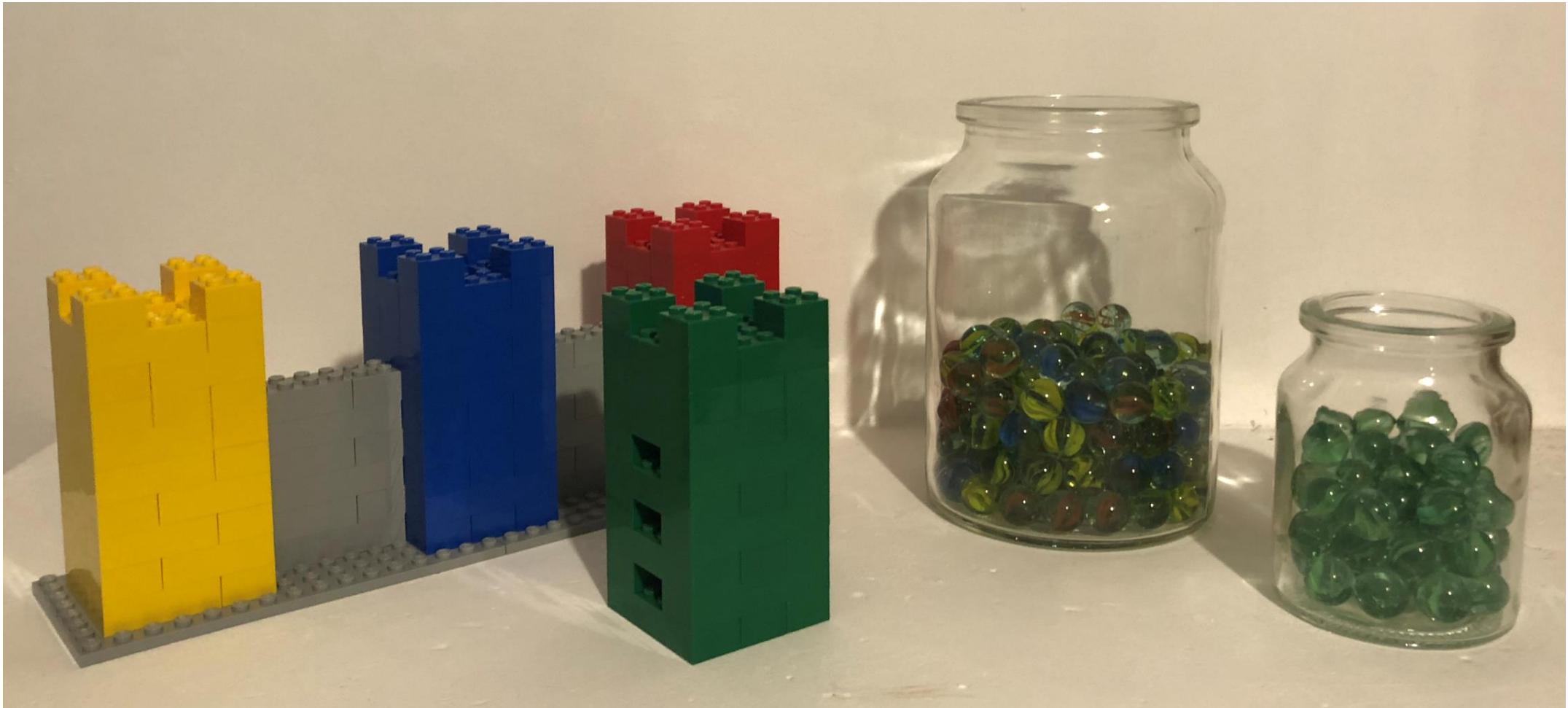
AXES

Step 4: SOLID

Axes



SOLID – A model for OO development



Axes

Step 5: Deployment models

- Plugins (in this talk)
- SOA
- Bus systems
- Microservices
- ...



The problem - revisited

What decisions did we make?

So, microservices, right?

You must be
this tall to use
microservices

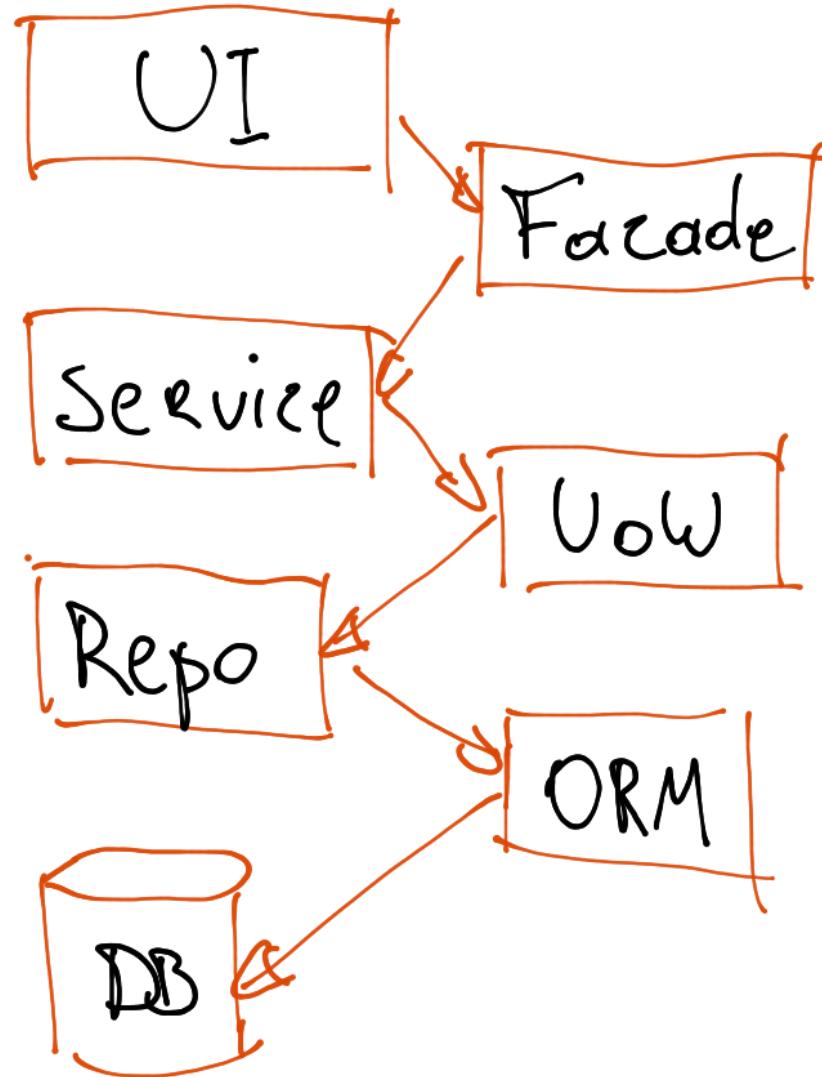


WRONG!

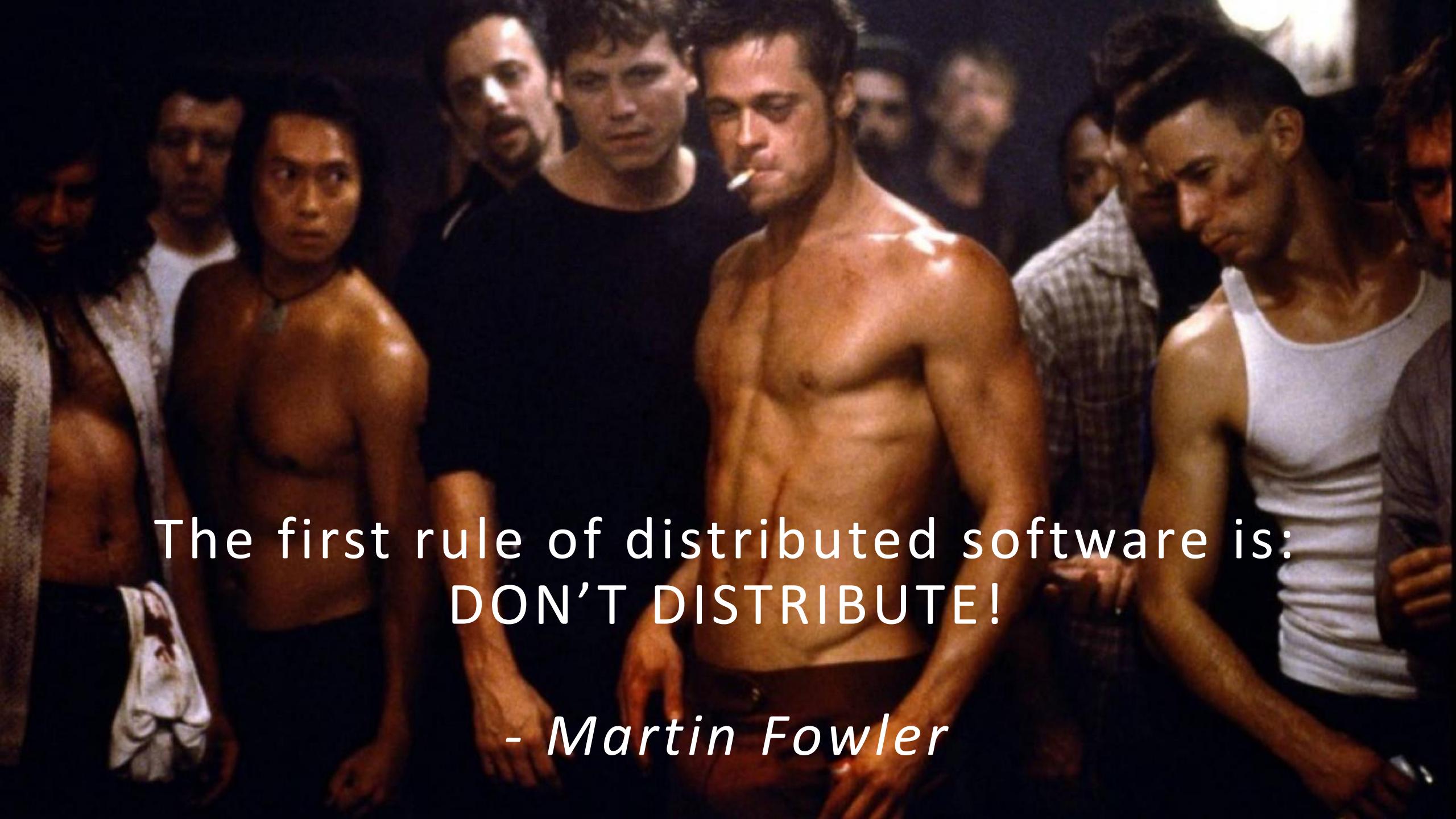
AXES

Step 3: "SOLID"

ABSTRACTIONS



AXES

A shirtless Brad Pitt stands in the center of a diverse crowd of men, looking down with a cigarette in his mouth. He is shirtless, showing a well-defined physique. The background is dark and filled with other men, some shirtless and some in casual clothing like tank tops and plaid shirts. The lighting is dramatic, highlighting Pitt's torso.

The first rule of distributed software is:
DON'T DISTRIBUTE!

- *Martin Fowler*



Never solve a code problem
by introducing a deployment problem!

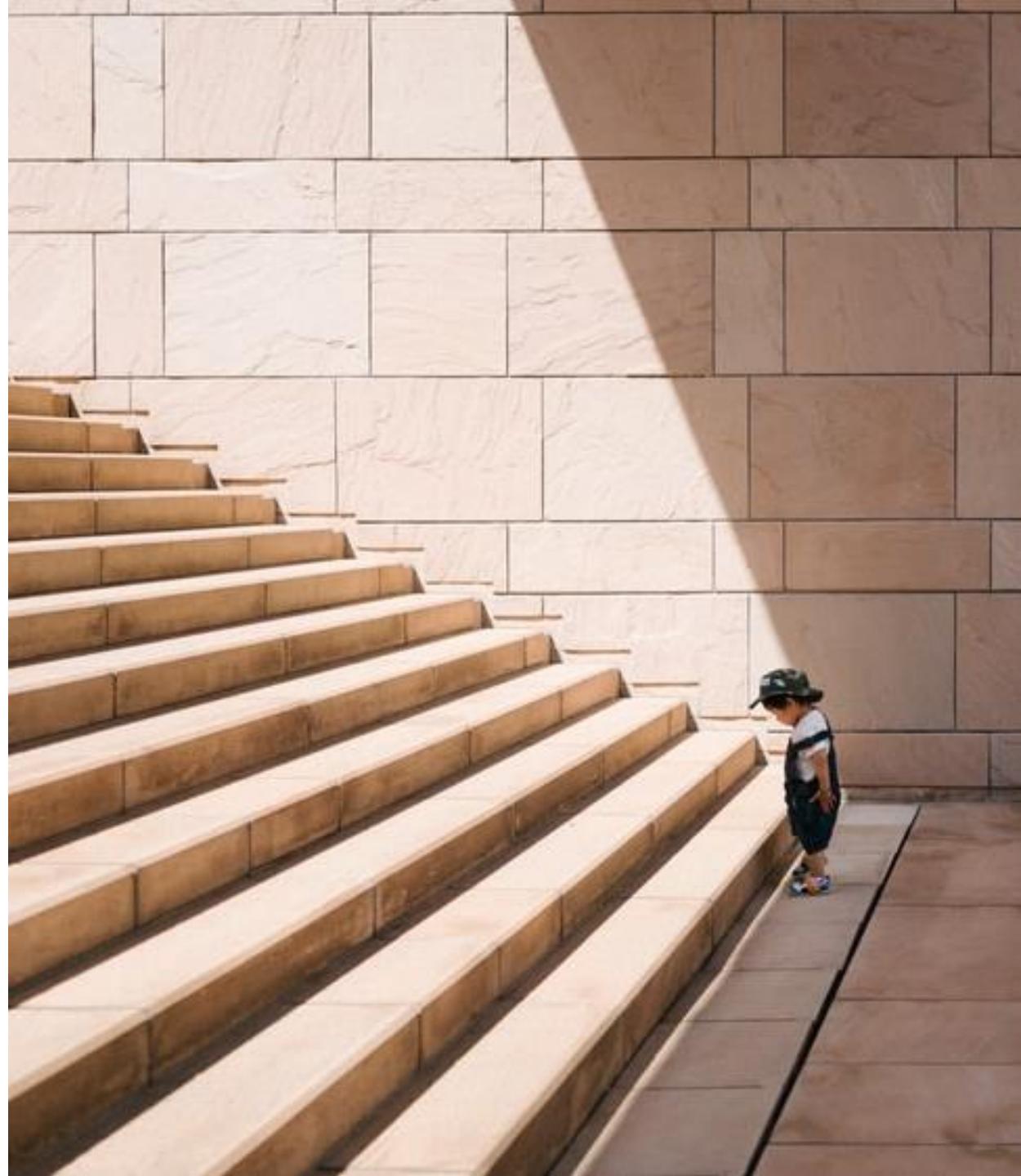
AXXES

- Me

Challenges

When adding a new device:

- Add controllers to ASP.NET application
- Extend API calls with derived types
→ *extend the central DbContext*
- Extend Logic in central API
- Handle DB Migrations



AXXES

What happened

- We picked: Plugins (in process)
- We built a successful POC
- We started doing this





Onion Architecture

—

SOLID-enabled solution architecture

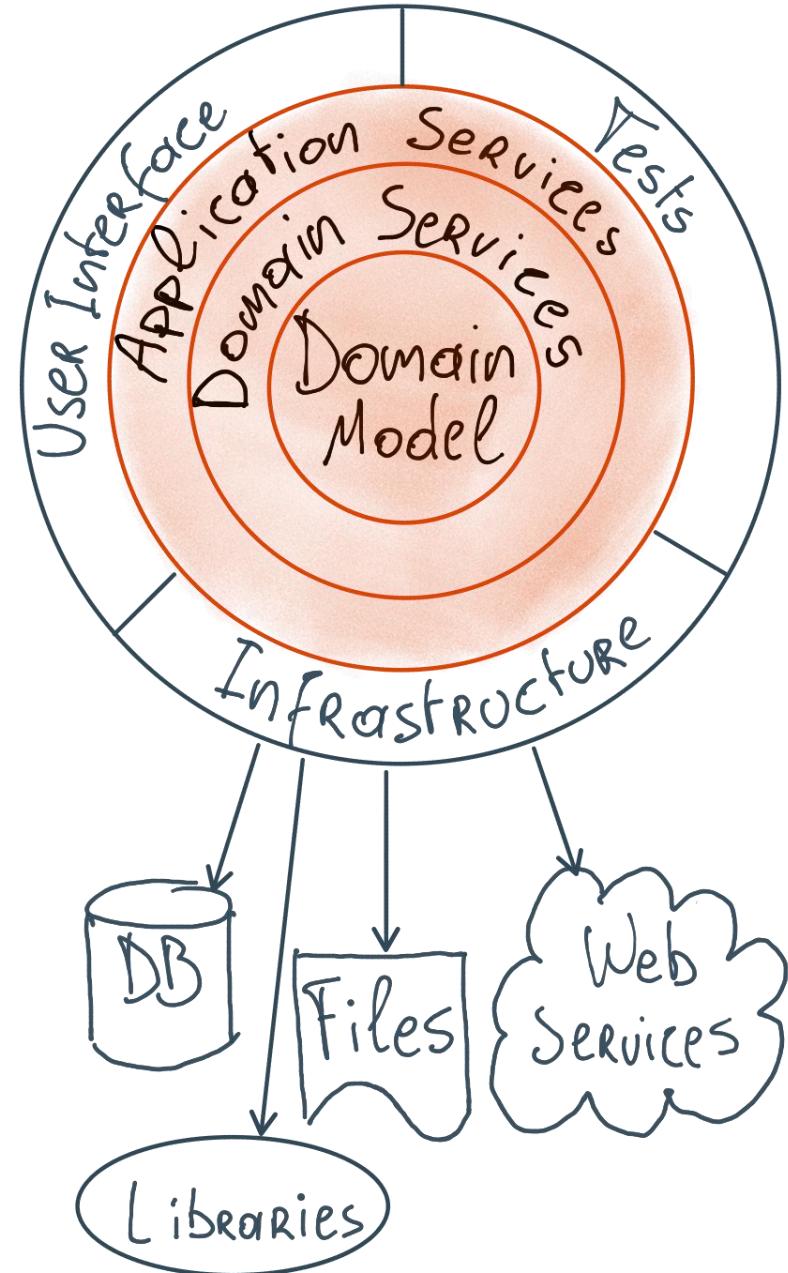
Onion Architecture

- Jeffrey Palermo in 2008
- Also known as:
 - ‘ports and adapters’
 - ‘hexagon architecture’
 - ‘clean architecture’
- Focus on:
 - Clean dependencies
 - Shielding abstractions
 - Testable business logic



Onion - Concepts

- References can only go 'in'
- The Core: reference free!
- Infrastructure for integrations:
 - The DB & ORM
 - File access & logging
 - External API calls
 - Libraries & packages!



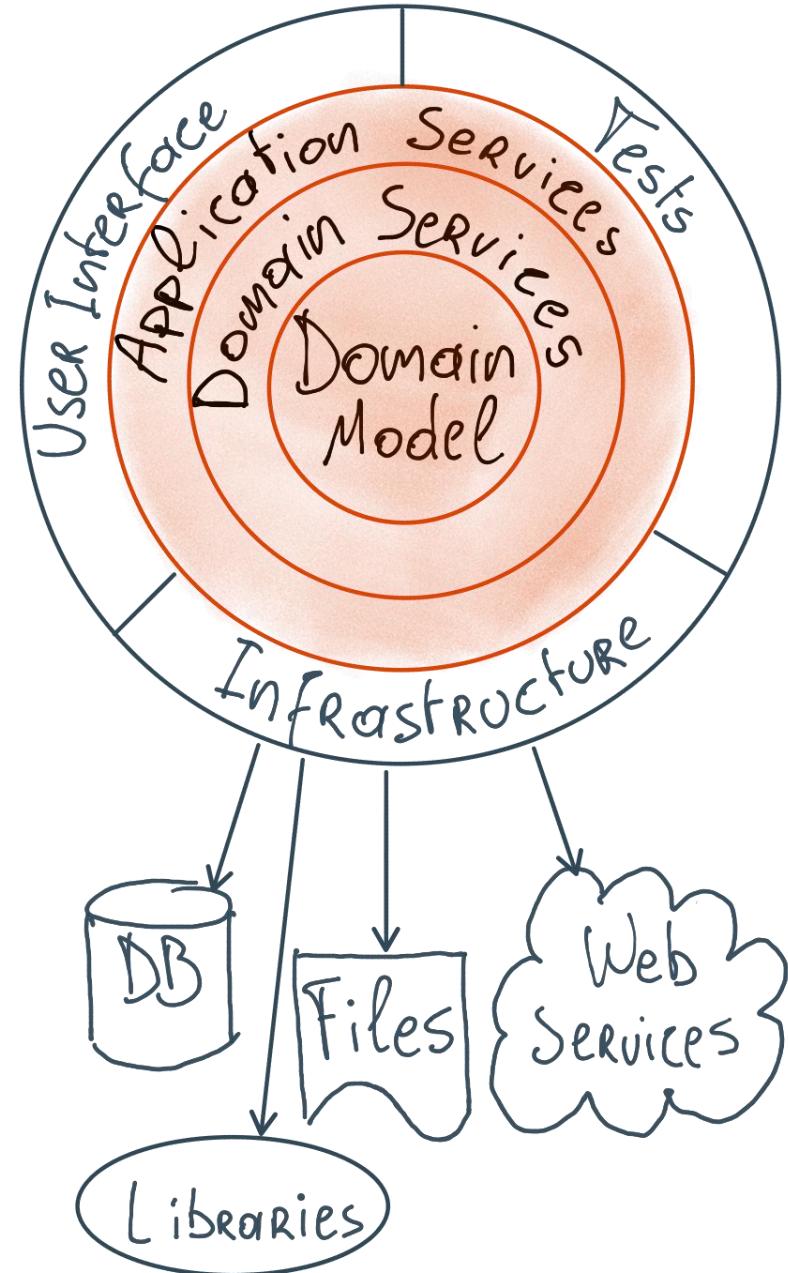
AXXES

Onion - Concepts

Benefits:

- No leaky dependencies
- Dependencies replaceable
- Reusable Core
- Forces you to write an interface first!
(interface owned by the consumer)

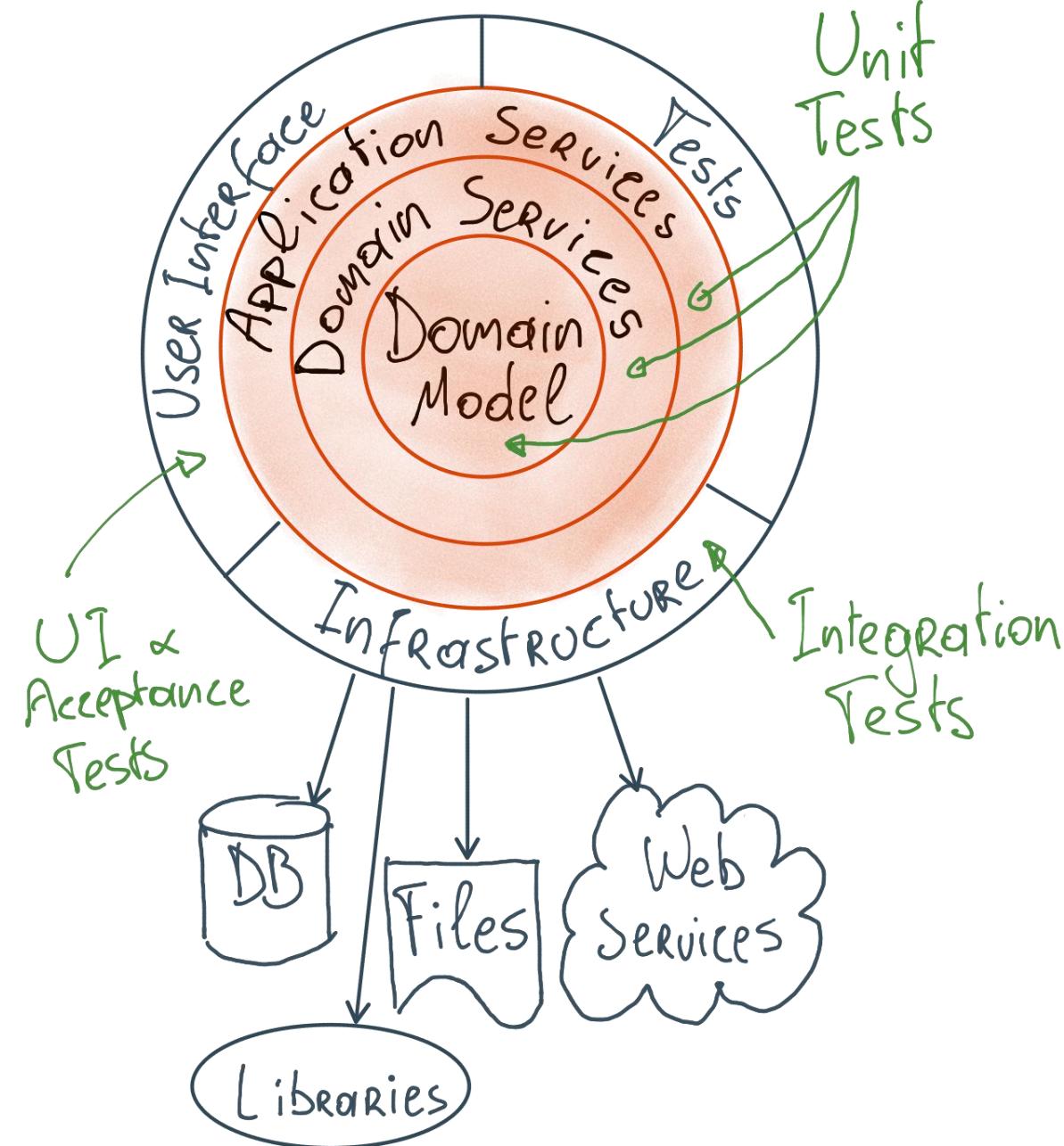
AXXES



Onion – Testing

Easy to:

- Test all individual components
- Determine the type of tests
- Define dependencies
- Mock dependencies



AXXES

Example: service locator

1. Define interface(s):

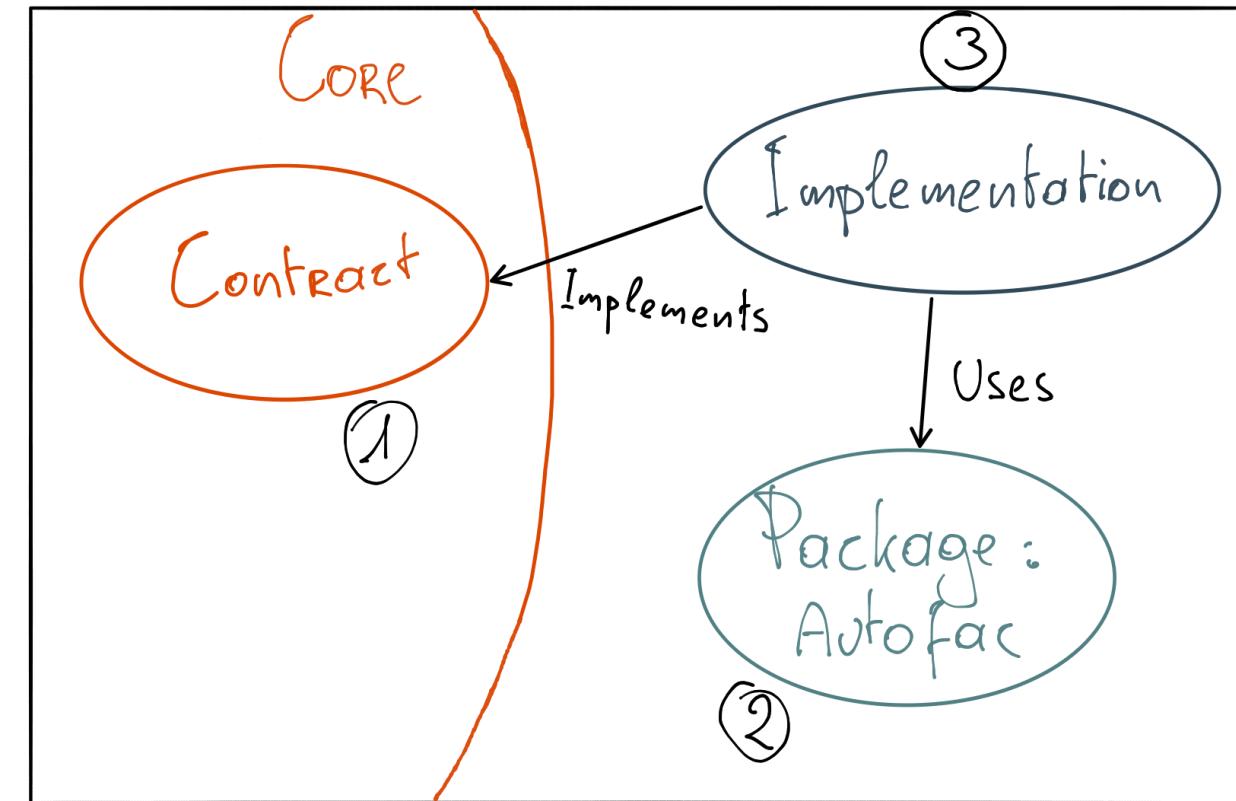
What do I need from a DI container?

2. Work smart:

Is there a package that fits this?

3. Write an implementation

Honestly, this should be trivial!



Integrations

The resulting integrations:

- Don't leak into your Core domain
- Easy to write
- Easy to test
- Easy to replace



Plugins

What do we expect from a plugin?

Plugin = assembly

- Easy to develop
- Extends our Core seamlessly
- Enable = ‘add the assemblies’
- Disable = ‘remove the assemblies’
- No references to the plugin from the Core!



AXXES

Plugin rules

- Can only reference the Core
- Should follow some conventions
- Can be deployed along with the application
- Should not break anything when removed



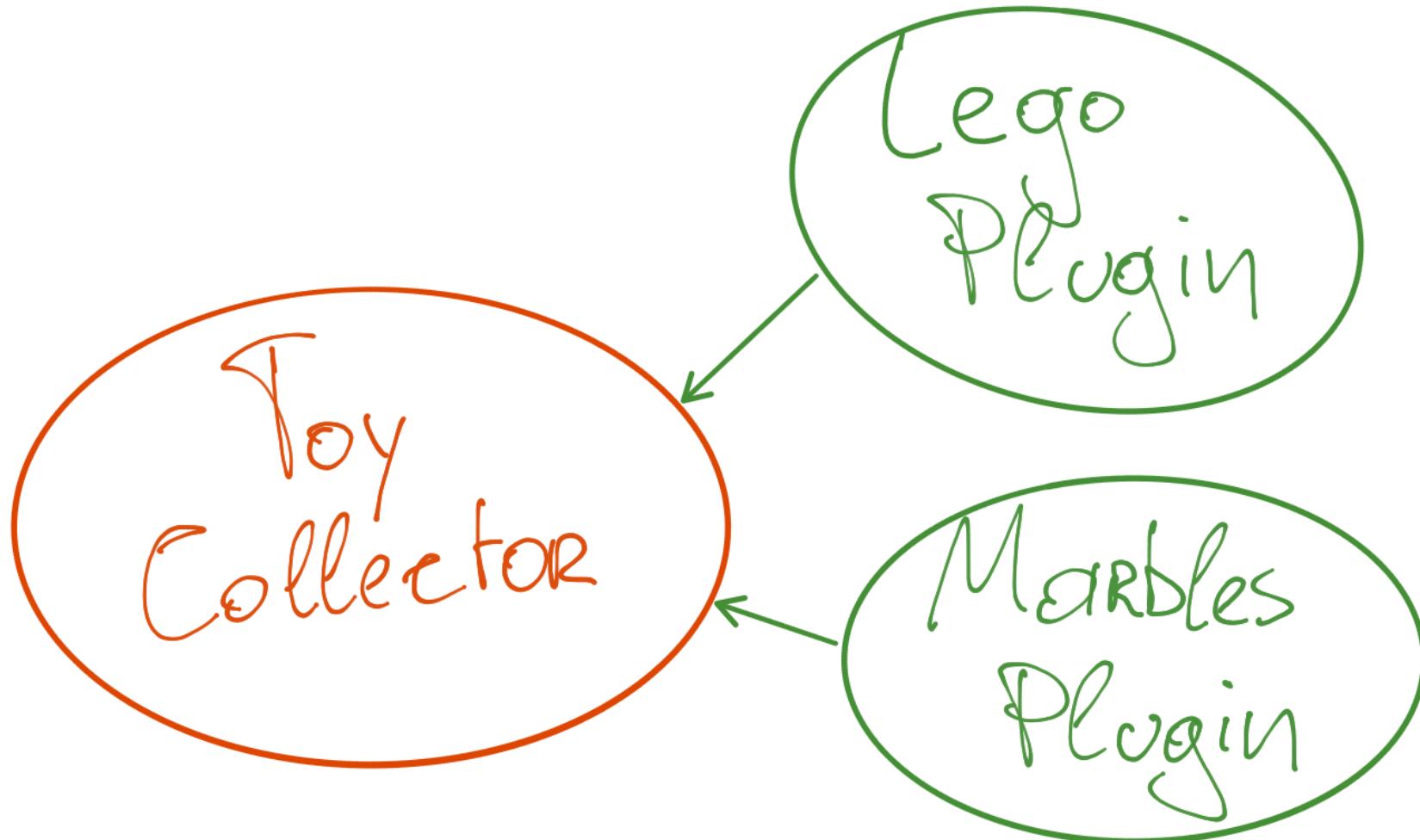
AXXES

Implementation

—

Enough chit-chat, show us some code!

Our example



AXES



What will we discuss?

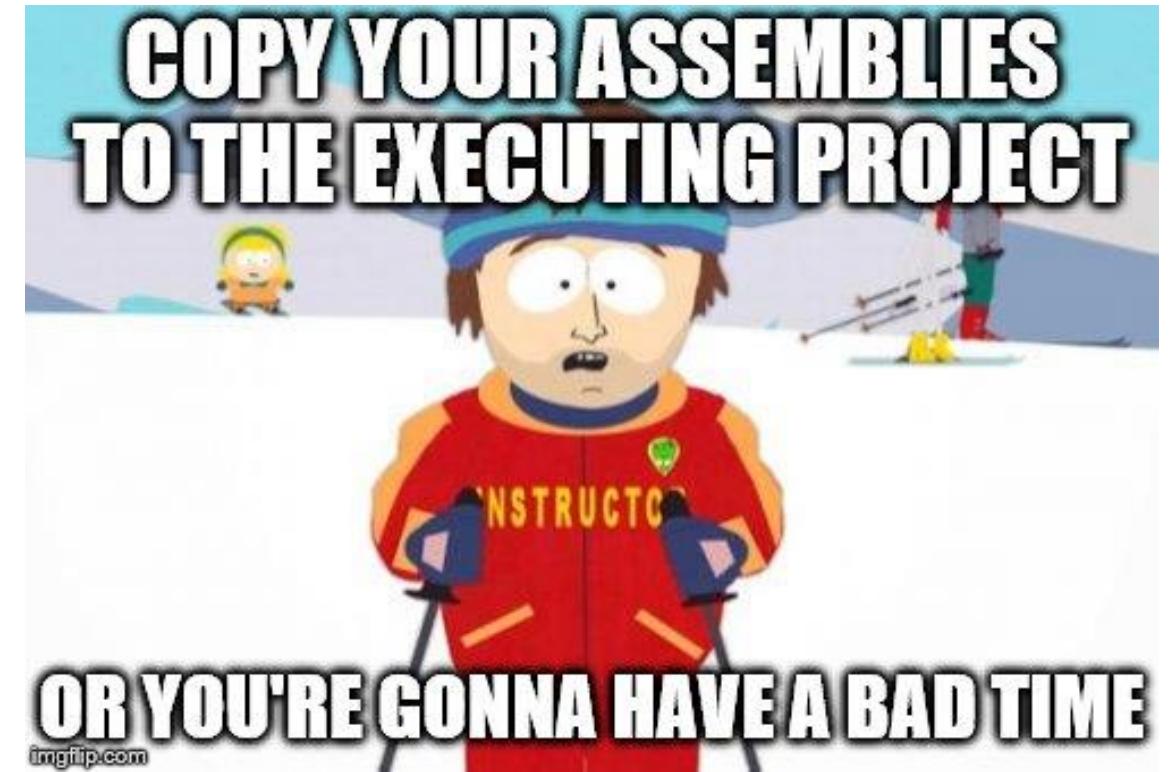
1. DI Container
2. ASP.NET controllers & views
3. JSON inheritance
4. Logic extension points
5. Extending Entity Framework
6. Migrations

Dealing with DI

- Scan assemblies at startup
- Use reflection
- Find our type registrar in each one
- Run the type registrars

IMPORTANT for development:

- .CSPROJ: copy in ‘post build actions’ (requires manual build)
- Make a development assembly with references



AXXES

```
private static void LoadAssembly(ITypeRegistrationContainer container, string dllFile)
{
    var assembly = Assembly.LoadFrom(dllFile); Just a wrapper around IServiceCollection

    var types = assembly.GetTypes();

    foreach (var registrarType in types
        .Where(t => typeof(ITypeRegistrar).IsAssignableFrom(t) && t.IsClass && !t.IsAbstract))
    {
        RunRegistrar(container, registrarType);
    }
}

private static void RunRegistrar(ITypeRegistrationContainer container, Type registrarType)
{
    var registrar = (ITypeRegistrar)Activator.CreateInstance(registrarType);

    registrar.RegisterServices(container);
}
```

AXES

ASP.NET

- **Controllers:**
Use the application part manager to add them
- **Views:**
Add the Views DLL (standard output in recent ASP.NET Core)
using `CompiledRazorAssemblyPart`
- **Pre-Core ASP.NET MVC:**
Custom Controller Selector & View Selector

AXES

```
private void LoadAspnetApplicationPlugins(ApplicationPartManager apm)
{
    var allPluginDlls = Directory.GetFiles(
        Path.Combine(Environment.ContentRootPath, "bin"), "Axxes.ToyCollector.Plugins.*.dll",
        SearchOption.AllDirectories);

    var allControllerDlls = allPluginDlls.Where(p => !p.EndsWith("Views.dll"));
    var allRazorViewsDlls = allPluginDlls.Where(p => p.EndsWith("Views.dll"));

    // Add MVC/API controllers from Plugins
    foreach (string controllerDll in allControllerDlls)
    {
        apm.ApplicationParts.Add(
            new AssemblyPart(Assembly.LoadFrom(controllerDll)));
    }

    // Add Razor views from Plugins
    foreach (string razorViewsDll in allRazorViewsDlls)
    {
        apm.ApplicationParts.Add(
            new CompiledRazorAssemblyPart(Assembly.LoadFrom(razorViewsDll)));
    }
}
```

AXXES

ASP.NET

- **Controllers:**
Use the application part manager to add them
- **Views:**
Add the Views DLL (standard output in recent ASP.NET Core)
using `CompiledRazorAssemblyPart`
- **Pre-Core ASP.NET MVC:**
Custom Controller Selector & View Selector

AXES

Posting inherited types

- Easy – XML:

```
<LegoSet>
  <description>Control Center</description>
  <acquireDate>1992-12-06T07:00:00.000Z</acquiredDate>
  <acquiredCondition>0</acquiredCondition>
  <currentCondition>2</currentCondition>
  <discontinuedDate>2002-07-01T00:00:00.000Z</discontinuedDate>
  ...
</LegoSet>
```

- Hard – JSON:

```
{
  "description": "Control Center",
  "acquiredDate": "1992-12-06T07:00:00.000Z",
  "acquiredCondition": 0,
  "currentCondition": 2,
  "discontinuedDate": "2002-07-01T00:00:00.000Z",
  "msrp": 175,
  "setNumber": "8094",
  "unopened": "false",
  "finishedBuildDate": "1992-12-07T21:00:00.000Z",
  "limitedEdition": "false"
}
```

JSON inheritance – the risky solution

- **Registration**

```
// Allows the passing of JSON $type parameters (required for inherited types)
mvcBuilder.AddJsonOptions(
    jsonOptions => jsonOptions.SerializerSettings.TypeNameHandling = TypeNameHandling.Auto);
```

- **Usage**

```
{
    "$type": "Axxes.ToyCollector.Plugins.Lego.Models.LegoSet, Axxes.ToyCollector.Plugins.Lego",
    "id": 0,
    "description": "string",
    "acquiredDate": "2018-11-12T08:10:17.177Z",
    "acquiredCondition": 0,
    "currentCondition": 0,
    "discontinuedDate": "2018-11-12T08:10:17.177Z",
    "msrp": 0
}
```

→ Serious vulnerability if you have an object/dynamic property

JSON inheritance – the proper solution

```
public class InheritedTypesJsonConverter : JsonConverter
{
    private const string TypePropertyName = "$type";
    private readonly InheritedTypesRegistry _inheritedTypesRegistry;

    public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
    {
        if (reader == null) throw new ArgumentNullException(nameof(reader));
        if (serializer == null) throw new ArgumentNullException(nameof(serializer));
        if (reader.TokenType == JsonToken.Null)
            return null;

        JObject jObject = JObject.Load(reader);

        var typeName = jObject[TypePropertyName]?.Value<string>();

        var target = _inheritedTypesRegistry.CreateType(objectType, typeName);

        serializer.Populate(jObject.CreateReader(), target);
        return target;
    }
}
```

Use this JSON property to detect the type
Inherited types are registered in this registry

We use the string value of the type\$ property
To construct an object of the proper type

JSON inheritance – the proper solution

- **Registration**

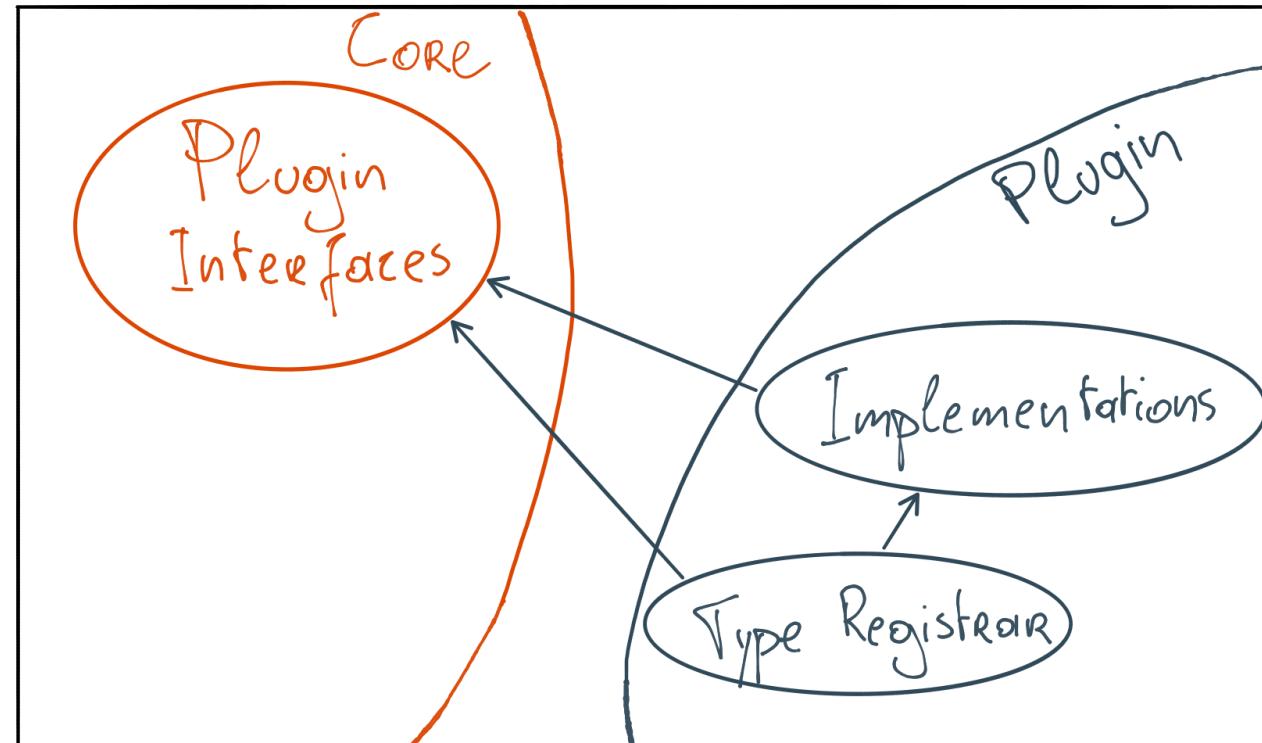
```
// Allows the passing of JSON $type parameters (required for inherited types)
mvcBuilder.AddJsonOptions(jsonOptions =>
{
    jsonOptions.SerializerSettings.Converters.Add(new InheritedTypesJsonConverter(inheritedTypesRegistry));
});
```

- **Usage**

```
{
    "$type": "LegoSet",
    "description": "Control Center",
    "acquiredDate": "1992-12-06T07:00:00.000Z",
    "acquiredCondition": 0,
    "currentCondition": 2,
    "discontinuedDate": "2002-07-01T00:00:00.000Z",
    "msrp": 175,
    "setNumber": "8094",
    "unopened": "false",
    "finishedBuildDate": "1992-12-07T21:00:00.000Z",
    "limitedEdition": "false"
}
```

Logic extension points

1. Define generic interfaces
2. Implement them in the plugin
3. Usage:
 - Scoped service locator
 - Simply inject?



```
public interface IToyCreatorCustomLogic
{
    void Execute(Toy newToy);
}

public interface IToyCreatorCustomLogic<T> : IToyCreatorCustomLogic
{
    where T: Toy
}
```

AXES

AXXES

```
public class ToyCreator : IToyCreator
{
    private readonly IToyRepository _repository;
    private readonly IScopeServiceLocator _serviceLocator;

    public ToyCreator(IToyRepository repository, IScopeServiceLocator serviceLocator)
    {
        _repository = repository;
        _serviceLocator = serviceLocator;
    }

    public async Task CreateToy(Toy toy)
    {
        await _repository.Create(toy);

        if (toy.GetType() != typeof(Toy))
        {
            RunCustomLogic(toy);
        }
    }

    private void RunCustomLogic(Toy toy)
    {
        var creatorInterfaceType = typeof(IToyCreatorCustomLogic<>);
        var toyType = toy.GetType(); Uses Reflection to construct the generic interface and resolve it
        var creator = _serviceLocator.ResolveGenericType(creatorInterfaceType, toyType);

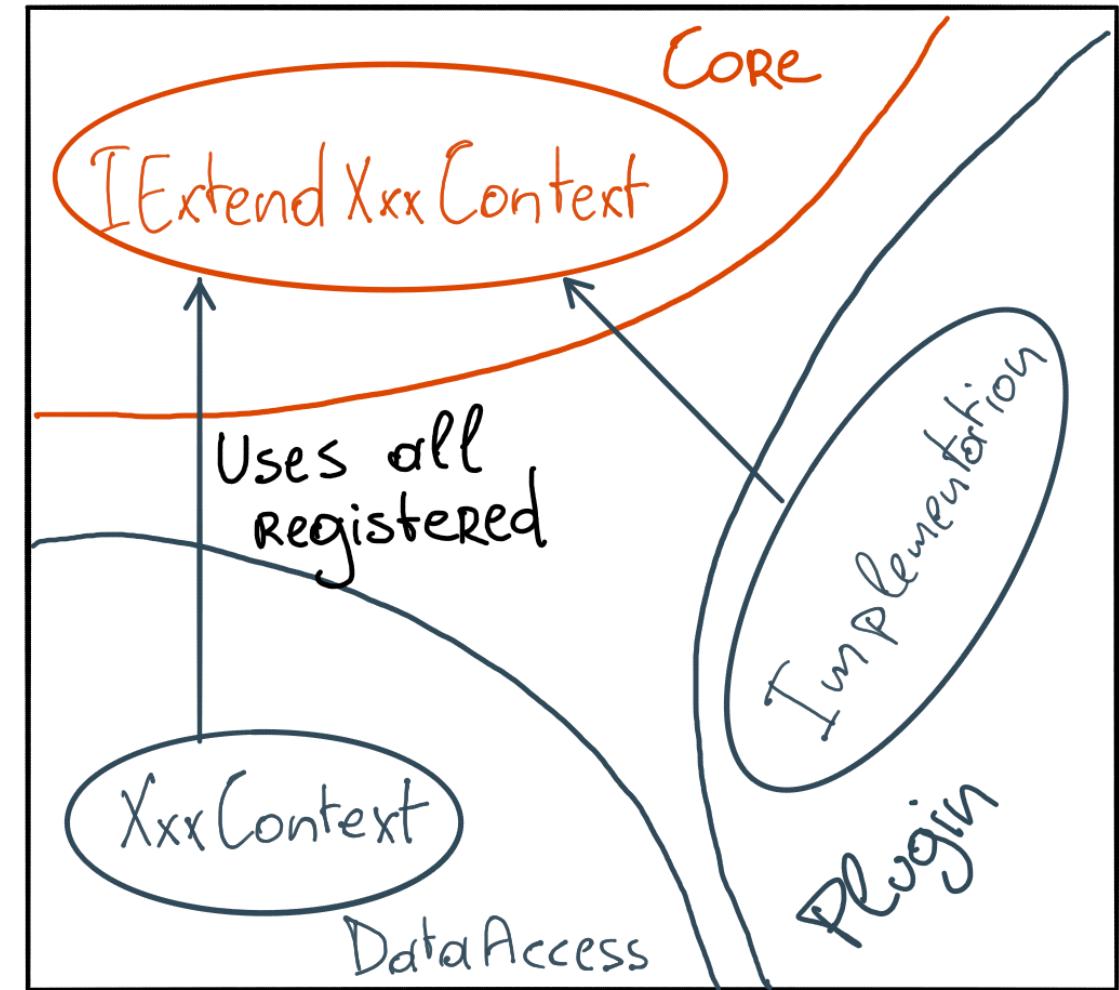
        if (creator != null && creator is IToyCreatorCustomLogic logic)
        {
            logic.Execute(toy);
        }
    }
}
```

Wrapper that resolves within the
current Scope (HttpRequest)

Uses Reflection to construct the generic interface and resolve it

Entity Framework

- Use `OnModelCreating` to feed the `DbContext` new (inherited) types
- EF Core = only TPH inheritance
- EF adds a Discriminator where clause



AXES

```
public interface IExtendToyContext
{
    void LoadToyContextExtensions(object builder);
}



---


public class ToyContextMableExtension : IExtendToyContext
{
    public void LoadToyContextExtensions(object builder)
    {
        if (builder is ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new MarbleMapping());
        }
    }
}
```

AXES

```
public class ToyContext : DbContext
{
    public ToyContext(
        IOptions<DatabaseConnectionStrings> connectionStrings,
        IEnumerable<IExtendToyContext> extensions)
    {
        _extensions = extensions;
        _connectionStrings = connectionStrings?.Value;
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new ToyMapping());

        foreach (var extension in _extensions)
        {
            extension.LoadToyContextExtensions(modelBuilder);
        }
    }
}
```

AXES

Database Migrations

Central Approach	Distributed approach
<ul style="list-style-type: none">+ Can be generated+ Easy to execute at deploy time	<ul style="list-style-type: none">+ Every plugin has its own migrations+ Database 100% in sync w/ plugins
<ul style="list-style-type: none">- All tables/fields exist even if the plugins are not loaded- Will require you to deal with the MigrationHistory table	<ul style="list-style-type: none">- Trickier to code & test- Requires runtime migrations

Central migrations: EF Core

- 1 Central Migrations project
- Easy to generate migrations!
- Uses service configuration of your startup project
BY DEFAULT
- **Without all plugins loaded, the model won't match!**
 - Generate migration scripts from the migrations
 - Rename the migration history table before/after deploy

Distributed migrations: FluentMigrator

- Every Plugin its own migrations
- Either:
 - Run migrations at runtime
 - Make a runner that loads the deployed plugins
- Migrations need to be hand-coded
- The FluentMigrator API is easy to learn
- No problems with the ModelState

```
[Migration(2018111200901)]
public class CreateToyTable : Migration
{
    public override void Up()
    {
        Create.Table("Toys").InSchema("dbo")
            .WithColumn("Id").AsInt32().PrimaryKey().Identity()
            .WithColumn("Description").AsAnsiString(250).Nullable()
            .WithColumn("Discriminator").AsString(int.MaxValue).NotNullable()
            .WithColumn("AcquiredDate").AsDate().NotNullable()
            .WithColumn("AcquiredCondition").AsInt32().NotNullable()
            .WithColumn("CurrentCondition").AsInt32().NotNullable()
            .WithColumn("DiscontinuedDate").AsDateTime2().Nullable()
            .WithColumn("Msrp").AsDecimal(18, 2).Nullable();
    }

    public override void Down()
    {
        Delete.Table("Toys").InSchema("dbo");
    }
}
```

AXES

```
public static class MigrationRunnerExtensions
{
    public static IMigrationRunnerBuilder ScanMigrations(this IMigrationRunnerBuilder builder,
        string[] pluginAssemblies)
    {
        // Core migrations
        var allFixedAssemblies = new[] { typeof(ToyContext).Assembly };

        // The plugins
        var allPluginAssemblies = pluginAssemblies.Select(Assembly.LoadFrom);

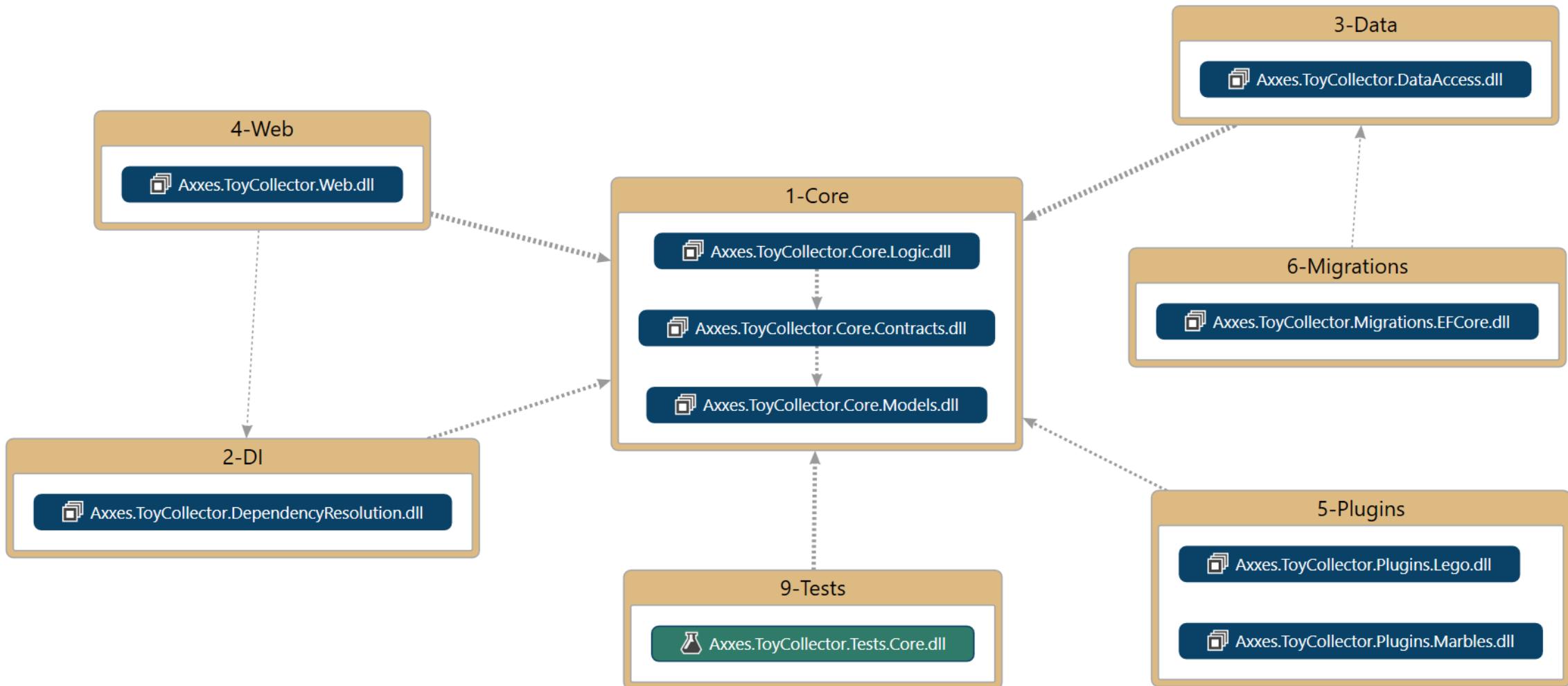
        var allMigrationAssemblies = allFixedAssemblies
            .Union(allPluginAssemblies)
            .ToArray();

        builder.ScanIn(allMigrationAssemblies).For.Migrations(); Needs to happen in ONE call

        return builder;
    }
}
```

AXES_

Project Dependencies



AXXES

Conclusion

Let's wrap up this session!

Key takeaways

- Before anything else, structure your code
- Plugins aren't too hard to do, especially in .NET Core
- Never solve a code problem by introducing a deployment problem!
- Don't worry if you're not at step 4, 5 or 8 yet.

Axes

FAQ

- Isn't this a lot harder?
- Can the plugins be Onions by themselves?
- When should I do this?

When should I use plugins?

- If you need modular deployments
(for instance: paid features per customer)
- If you want to easily retire/replace features
- If you want to be able to test features (A/B)
- To make smaller build pipelines in a large product

And if your business OK's the 10/100 rule ...

Questions?



AXXES
IT CONSULTANCY



About me

Hannes Lowette

Head of Learning & Development at @Axxes_IT

 @hannes_lowette

 #20086521

Code samples and slides at :

<https://axxes.com/dotnet/marbles-not-lego/>



Thank you!

Entrepotkaai 10A,
2000 Antwerpen

Leonardo Da Vincielaan 9,
1930 Zaventem

Ottergemsesteenweg Zuid 808
bus 300 , 9000 Gent

T +32 3 23499.58
info@axxes.com

www.axxes.com
—



Images from

<https://www.habausa.com/>

<http://technicopedia.com/>

<https://bricks.stackexchange.com/>

<https://tenor.com/>

<https://www.reddit.com/>

<https://pixabay.com/>

<https://martinfowler.com/>

<https://wastelessfuture.com/>

<https://me.me/>

<https://buildplease.com/>

<https://noisebreak.com/>

<https://www.theonion.com/>

<https://www.psychologicalscience.org/>

<http://www.gillsonions.com/>

<https://www.toyotaofplano.com/>

<https://imgflip.com/>

<https://www.asp.net/>

<https://www.amazon.com/>

<https://xkcd.com/>

<https://unsplash.com/>