AXXES

Hannes Lowette

Father of Arne (12), Joren (8) & Marit (6)

Partner of Barbara (?)

Head of L&D @ Axxes

.NET backend dev

Loves knowledge sharing

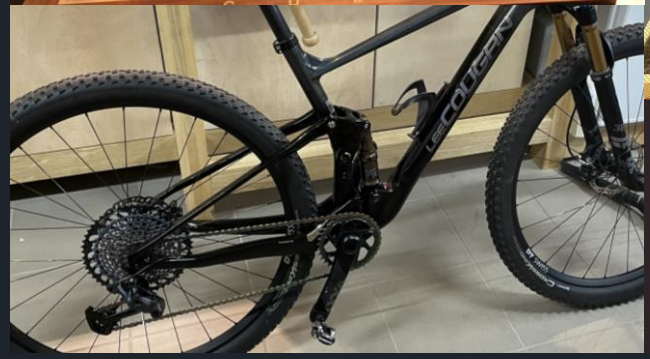
Amateur luthier (guitar builder)

Guitarist @ Dylan Beattie & the Linebreakers

Mountain biker

Average chess player

Microsoft MVP



AXXES

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’

AXXES

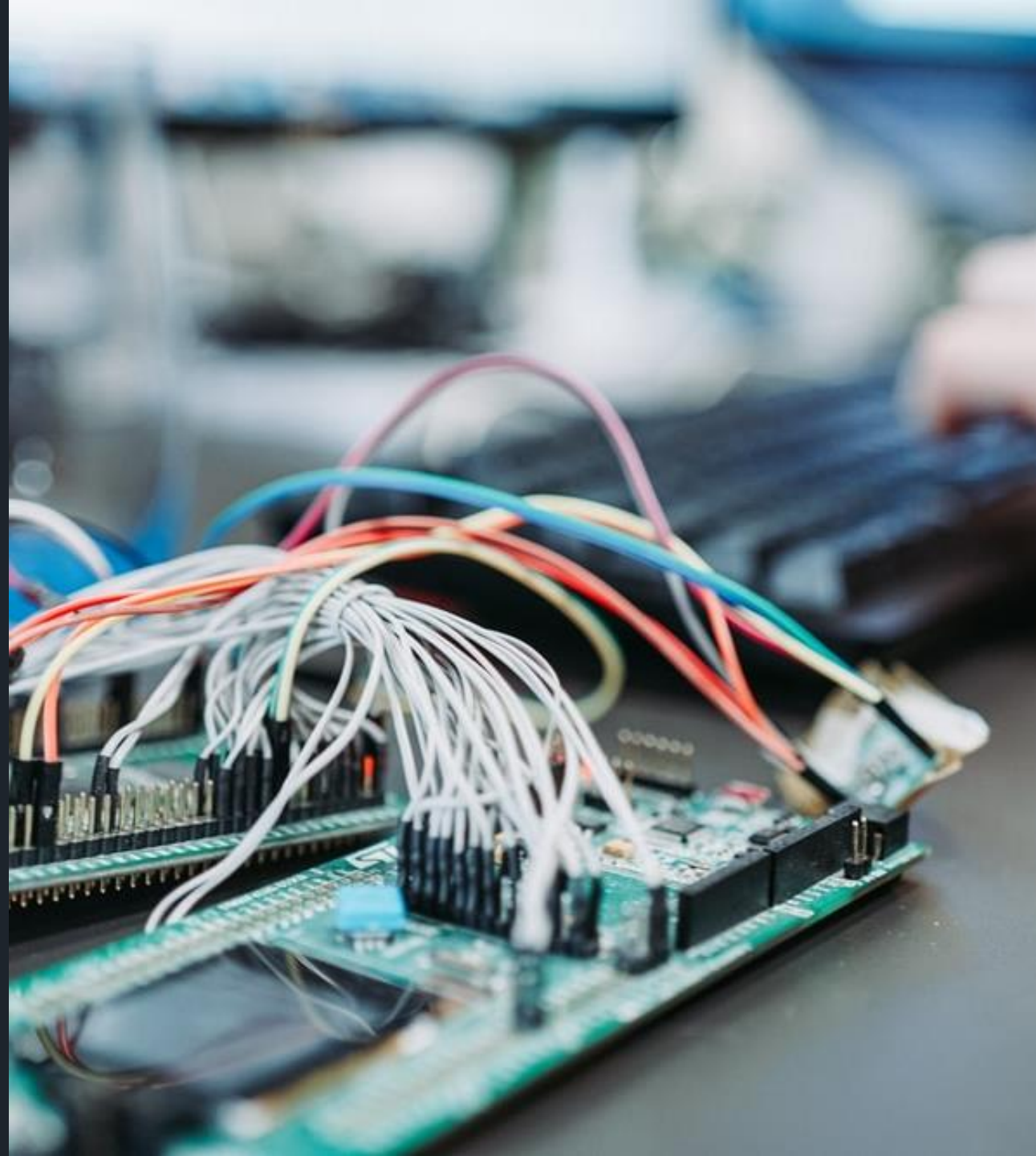
Understanding the problem

–

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



AXXES

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!

AXXES

What frustrated me about this?

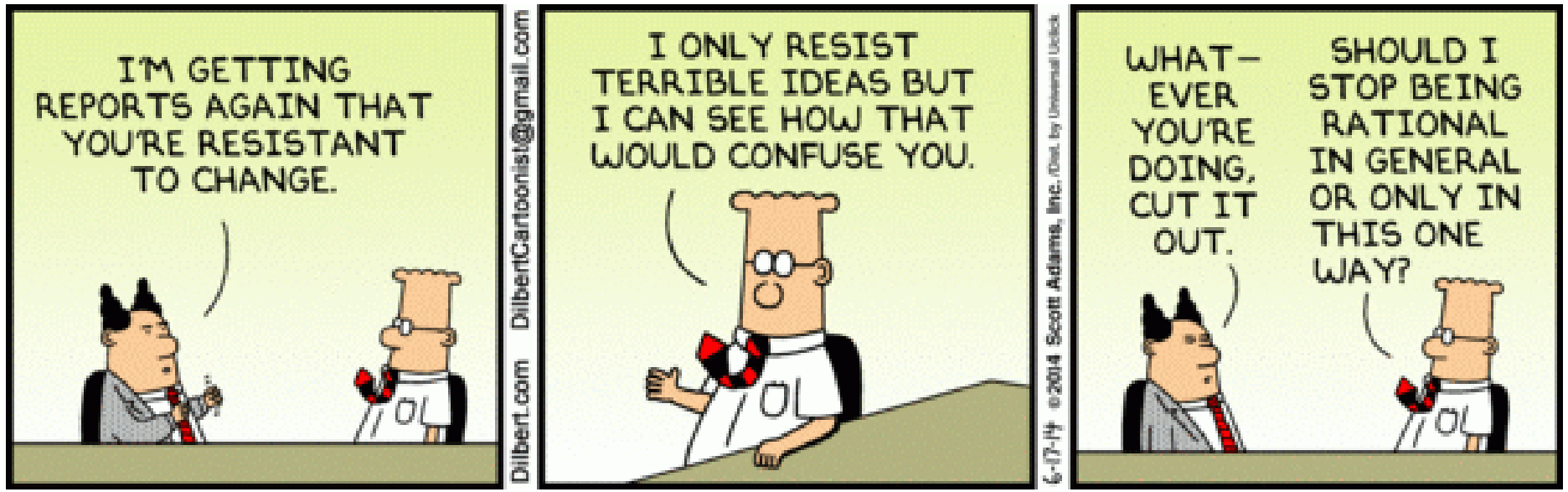
One of my most productive days was throwing away 1,000 lines of code.

Ken Thompson

AXXES

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?

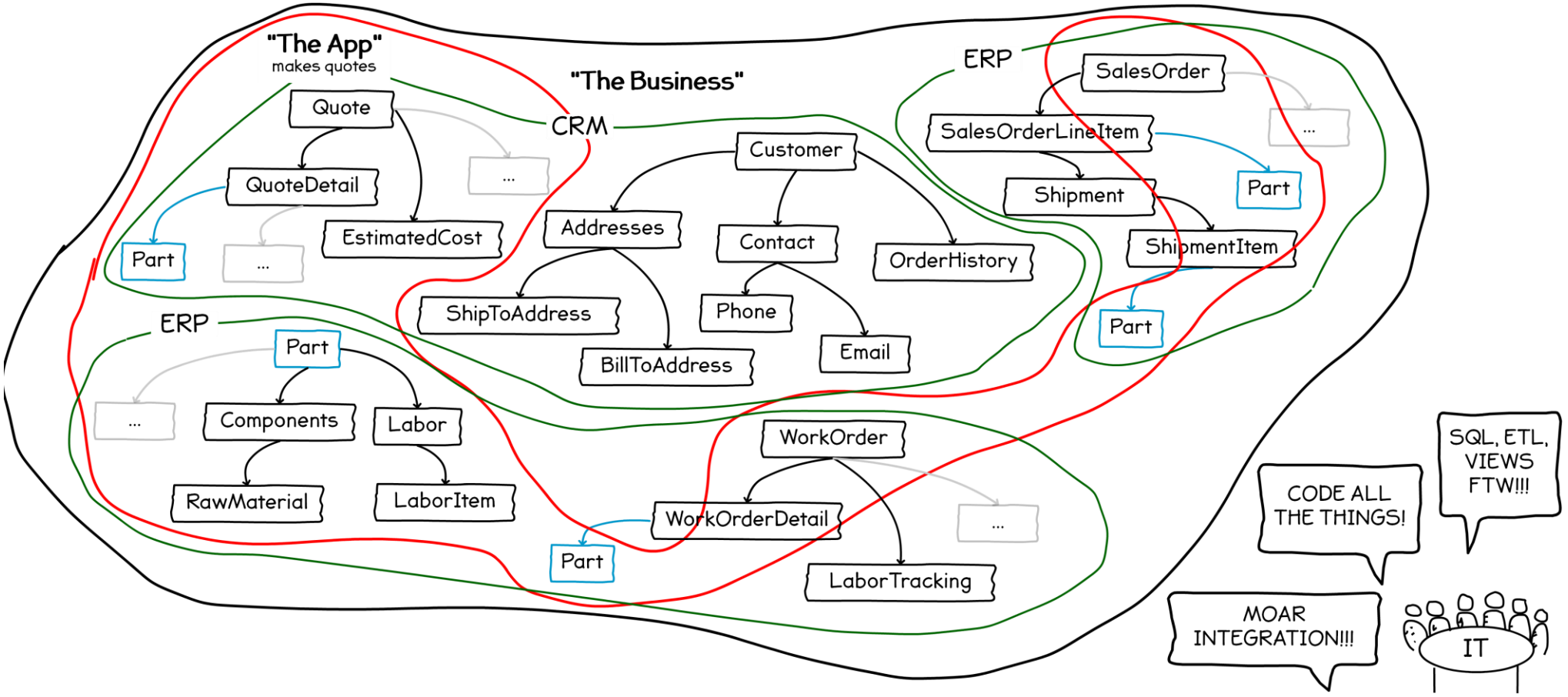
AXXES

The OO path to success

–

Steps in our evolutions as developers

Step 1: Demoware

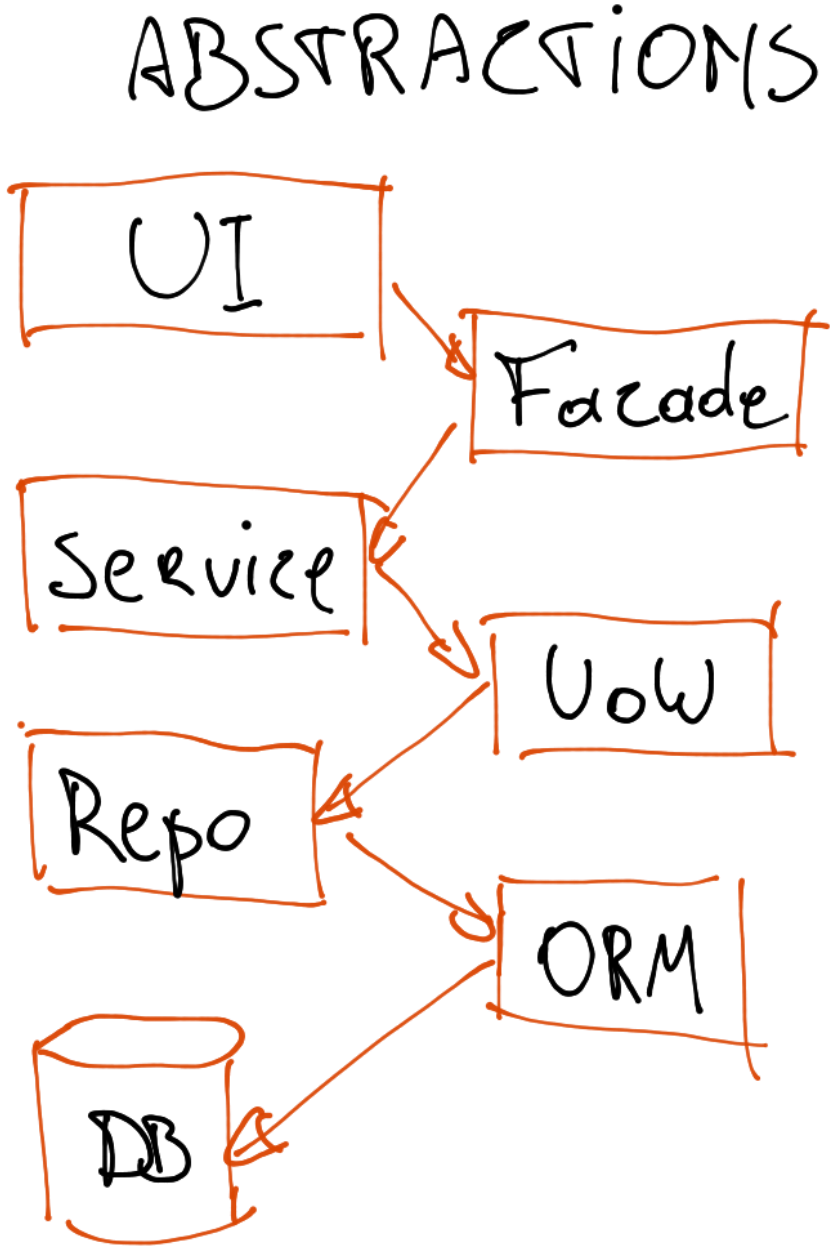


AXXES

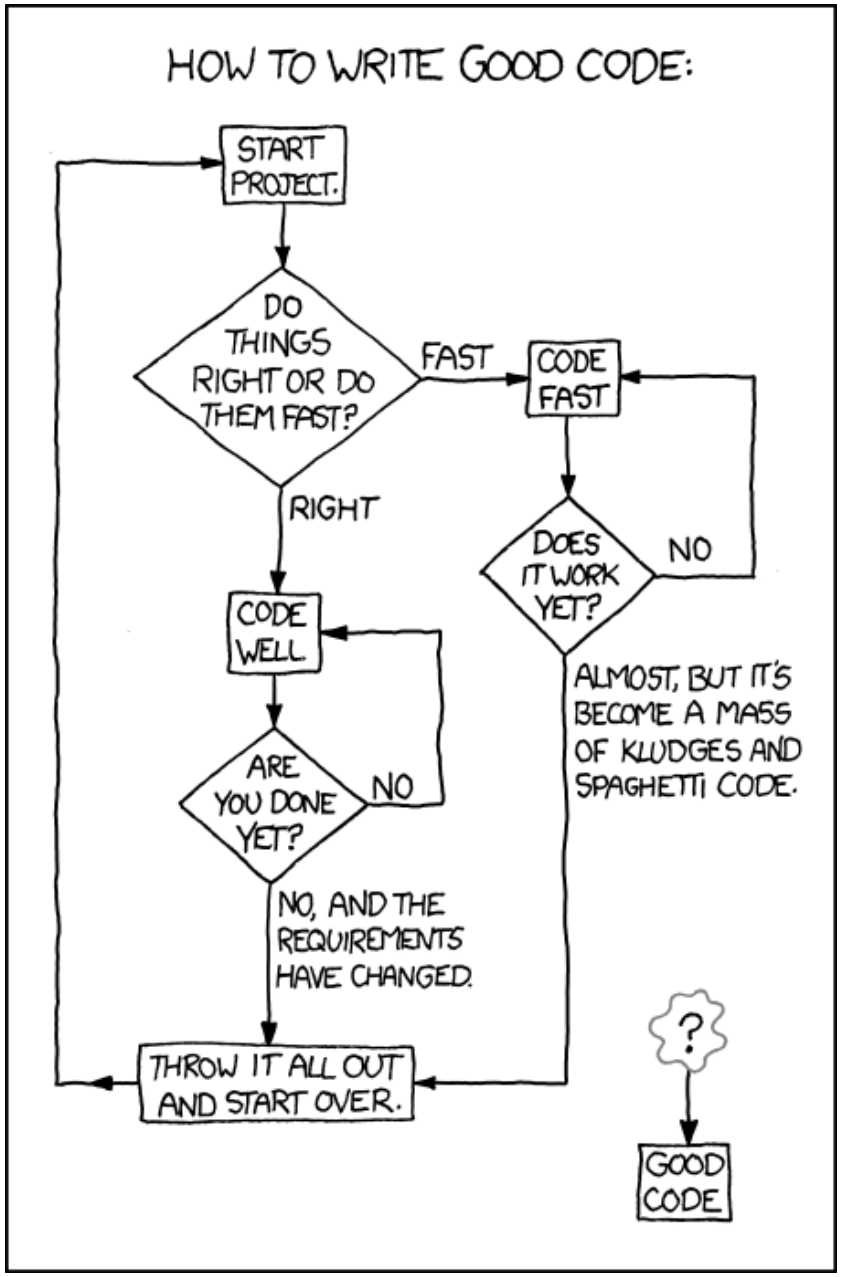
Step 2: Layers

AXXES

Step 3: “SOLID”

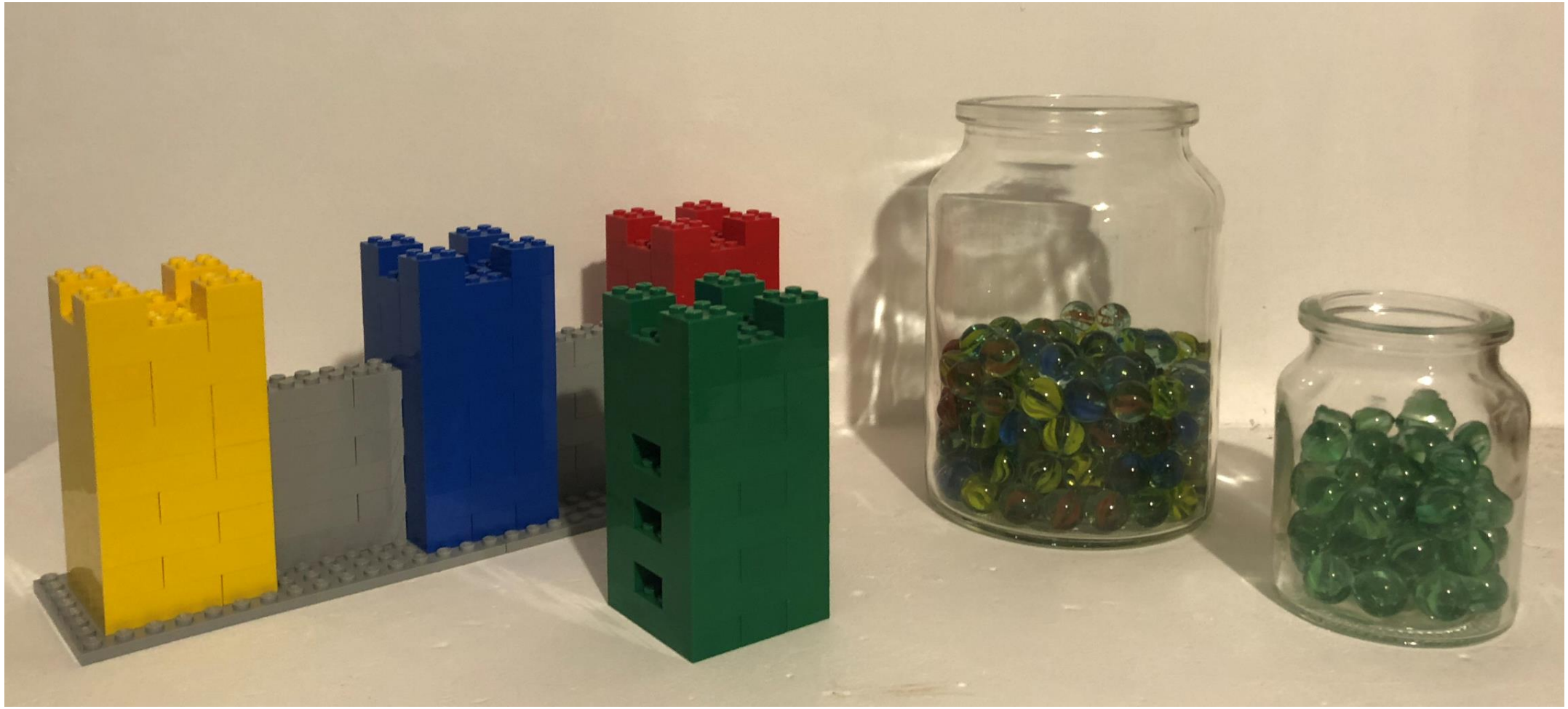


Step 4: SOLID



AXXES

SOLID – A model for OO development



AXXES

Step 5: Deployment models

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

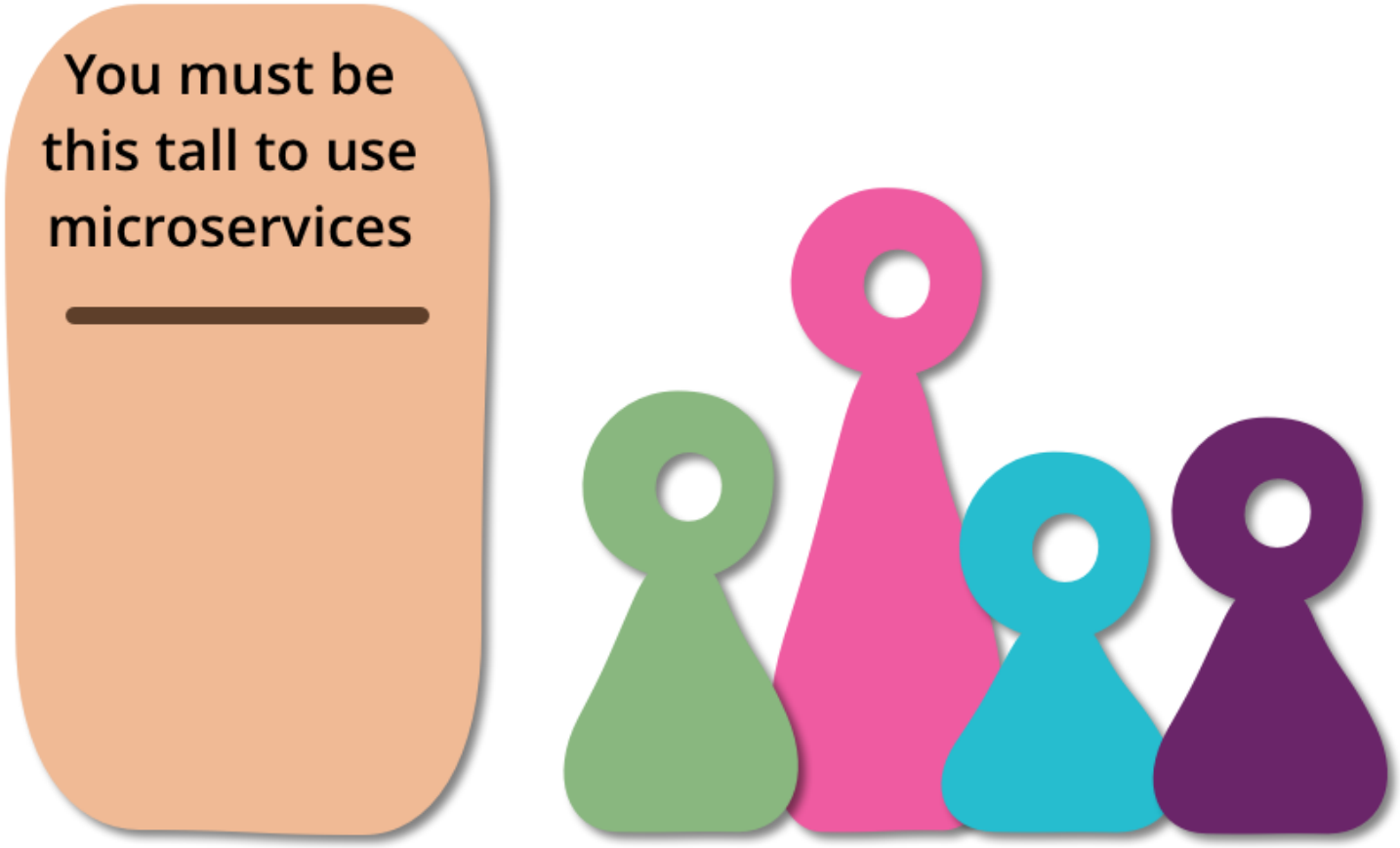
AXXES

The problem - revisited

–

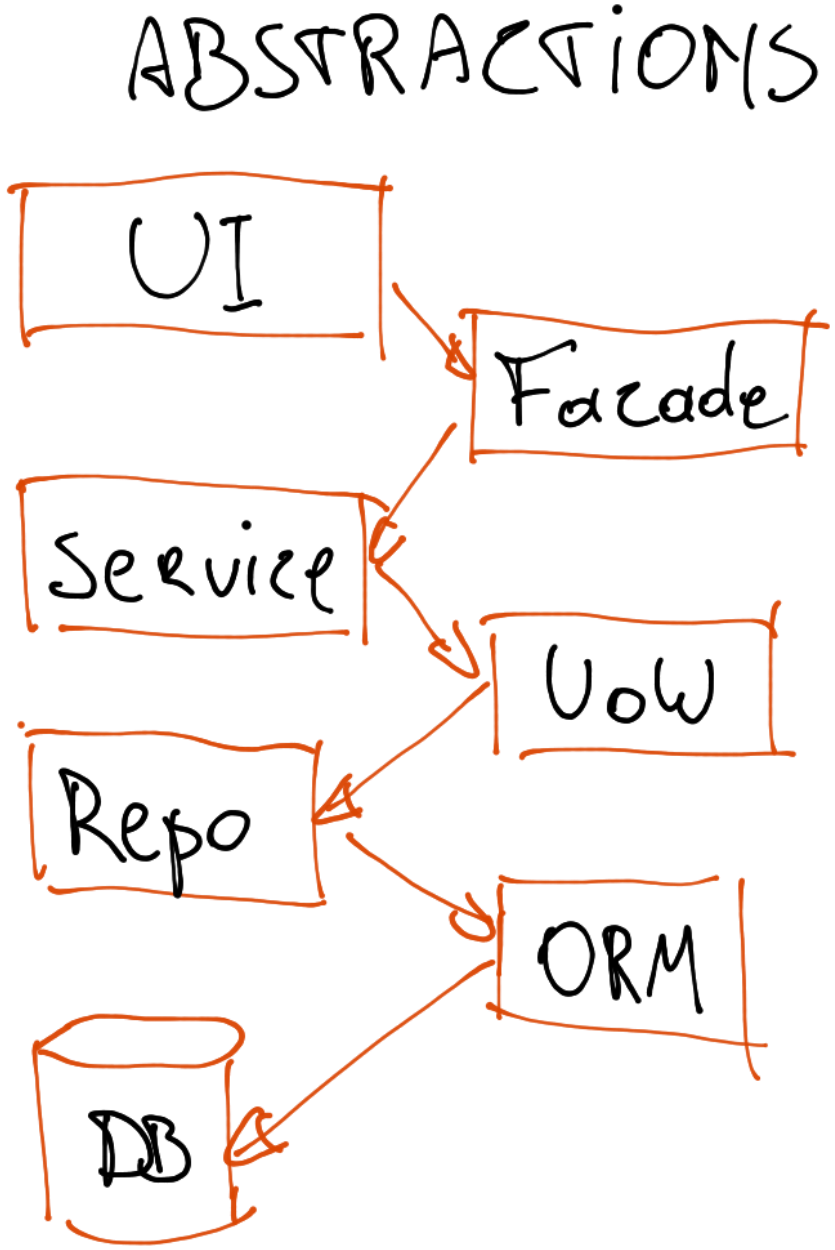
What decisions did we make?

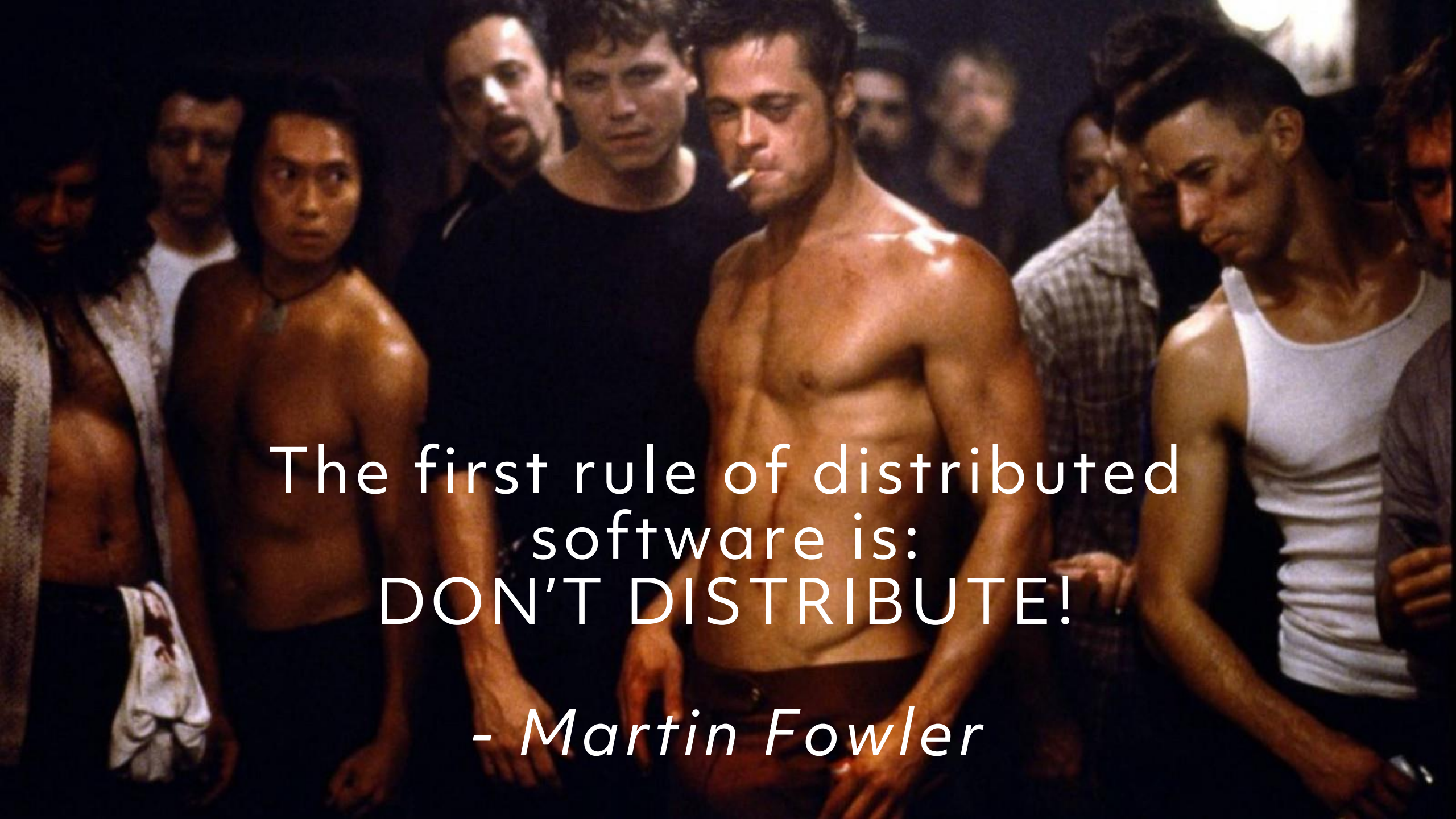
So, microservices, right?



WRONG!

Step 3: “SOLID”




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

Never solve a code problem
by introducing a deployment problem!

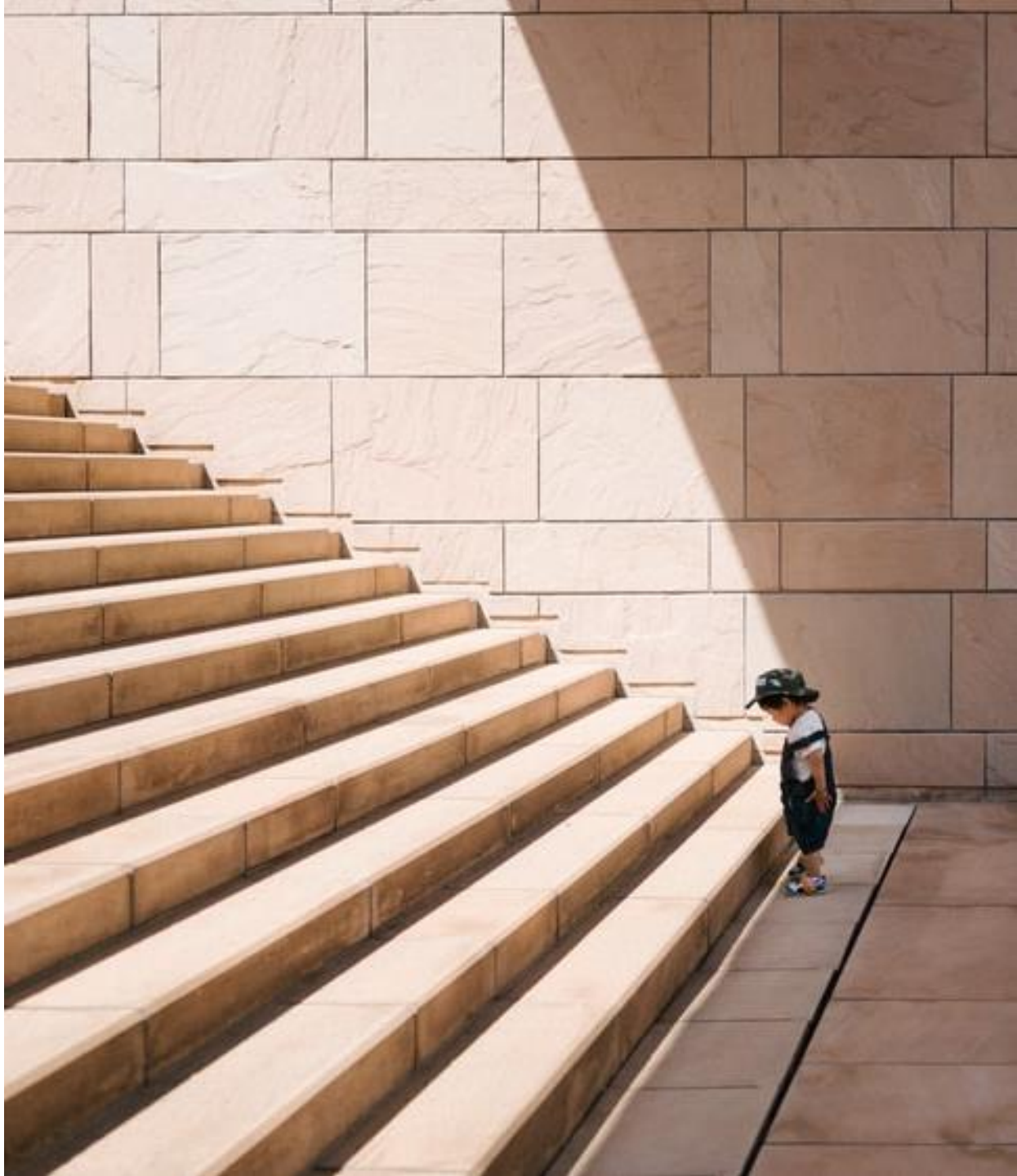
- *Me*

AXXES

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

AXXES

Onion Architecture

–

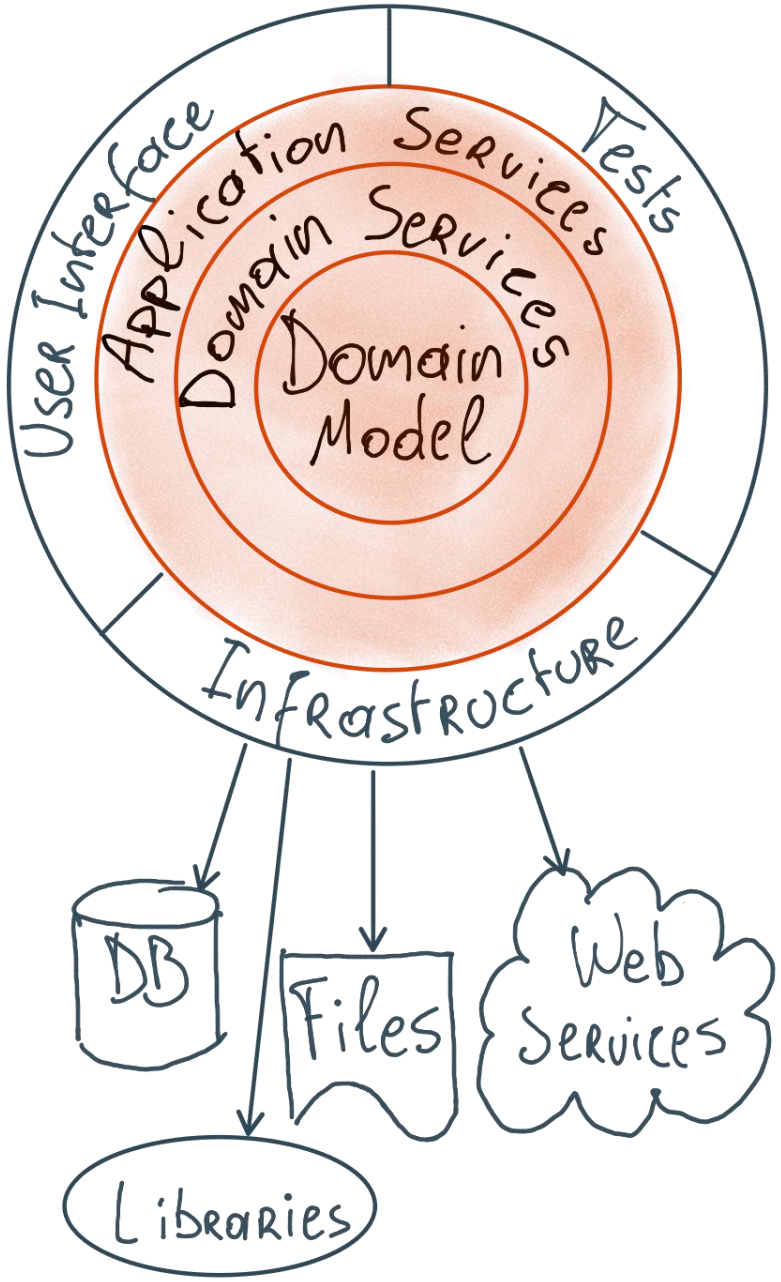
SOLID-enabled solution architecture

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!

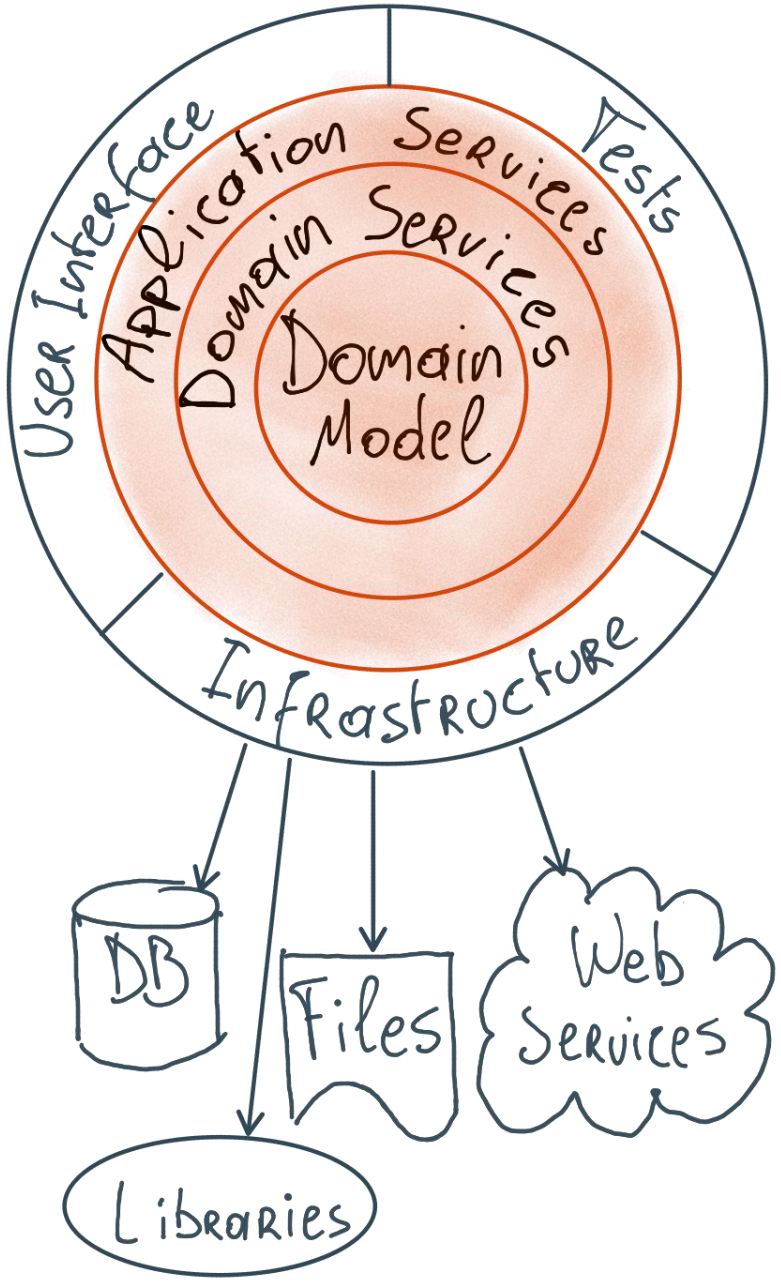


AXXES

Onion - Concepts

Benefits:

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

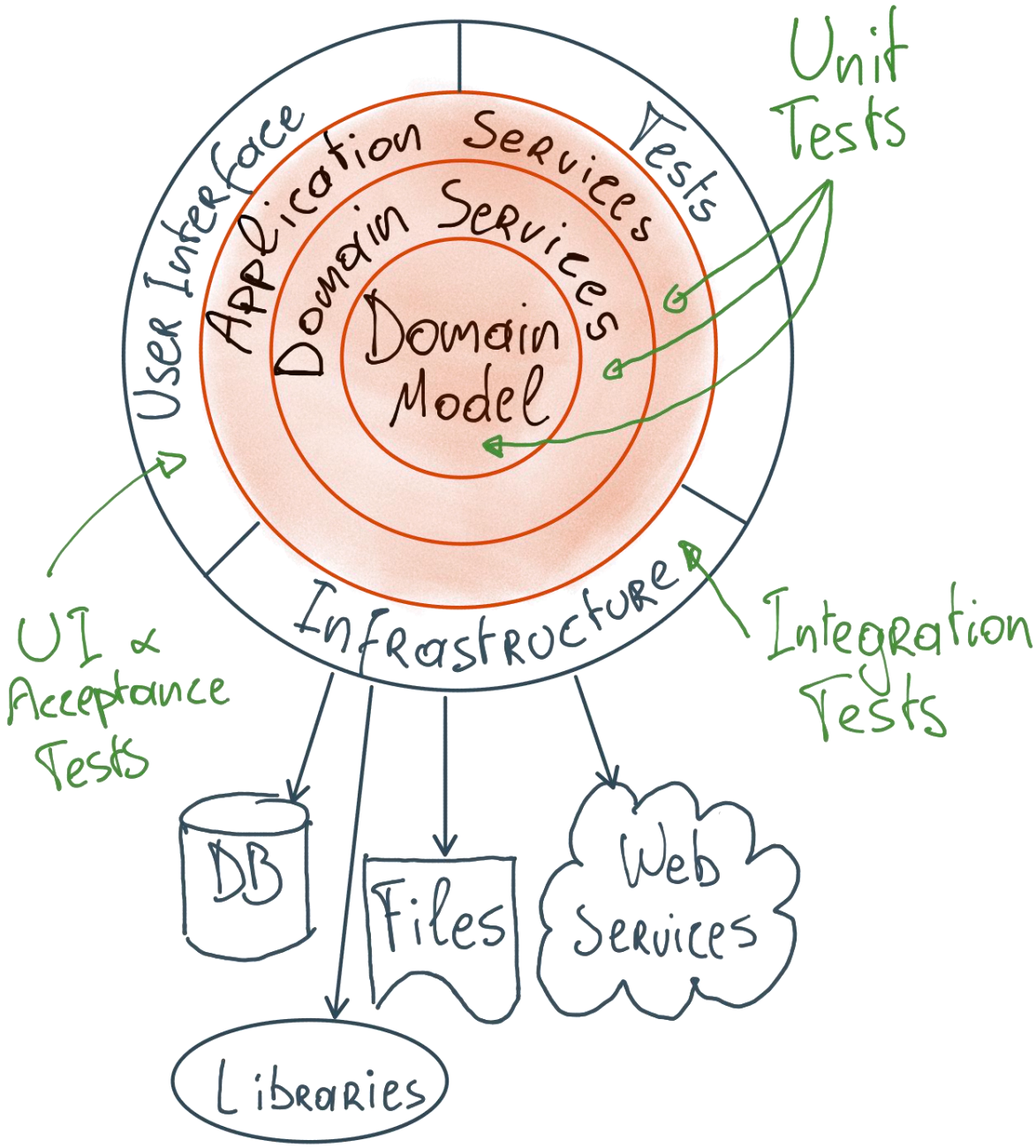


AXXES

Onion – Testing

Easy to:

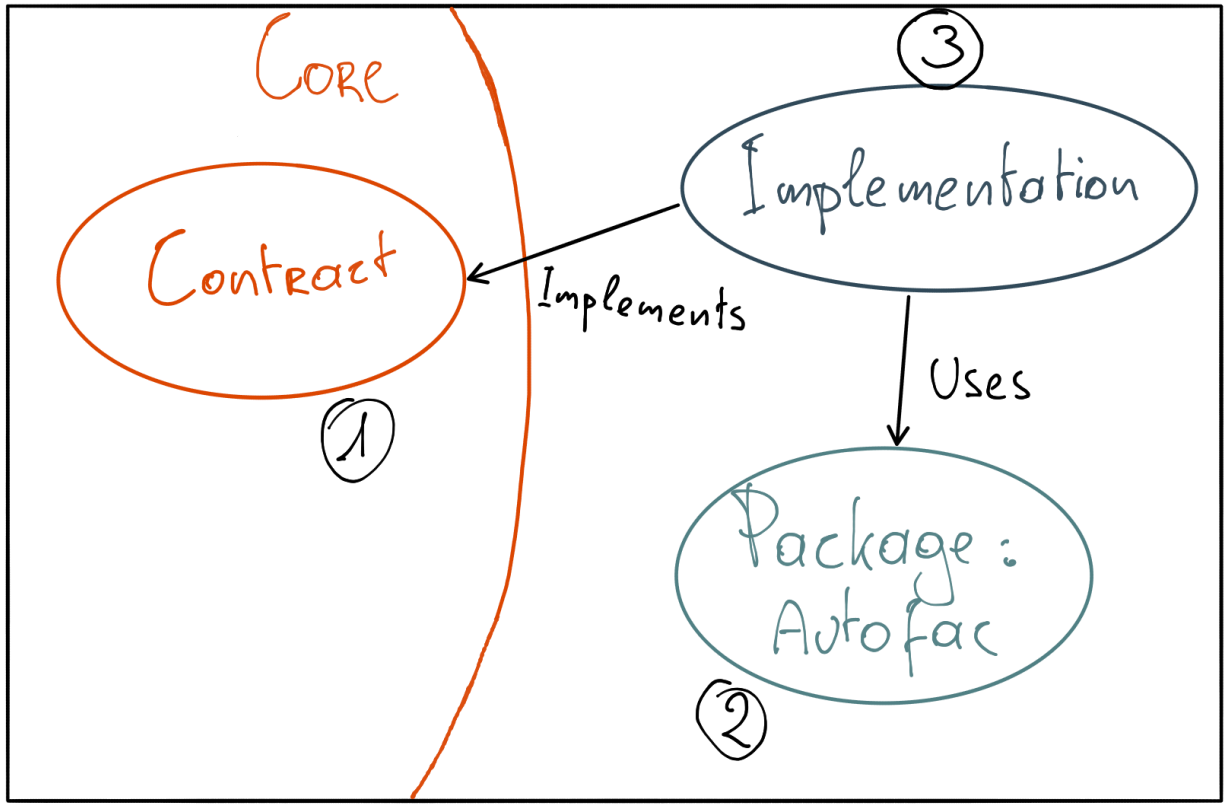
- Test all individual components
- Determine the type of tests
- Define dependencies
- Mock dependencies



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
   Use it to implement the interface.



Integrations

The resulting integrations:

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

AXXES

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 with the applicatio
- Don't break anything when removed

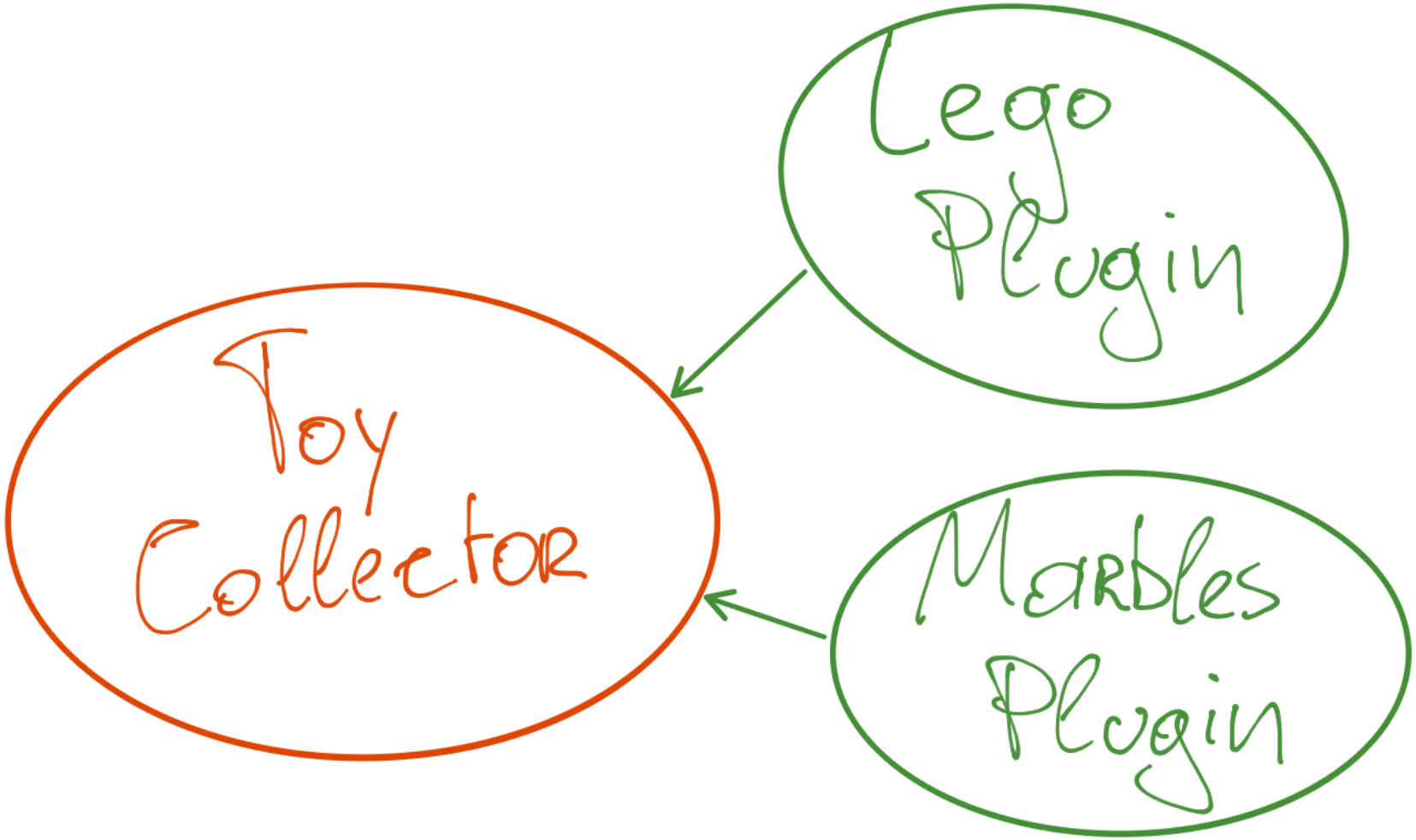
AXXES

Implementation

–

Enough chit-chat, show us some code!

Our example



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

AXXES

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

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

    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);
}
```

Just a wrapper around IServiceCollection

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

AXXES

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


    foreach (string pluginDll in allPluginDlls)
    {
        var assembly = Assembly.LoadFrom(pluginDll);
        // Add MVC/API controllers from Plugins
        apm.ApplicationParts.Add(new AssemblyPart(assembly));
        // Add Razor views from Plugins
        apm.ApplicationParts.Add(new CompiledRazorAssemblyPart(assembly));
    }
}
```

AXXES

Posting inherited types

- Easy – XML:

```xml
<LegoSet>
    <descrition>Control Center</descrition>
    <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:

```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"
}
```

AXXES

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

AXXES

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

AXXES

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));
});
```

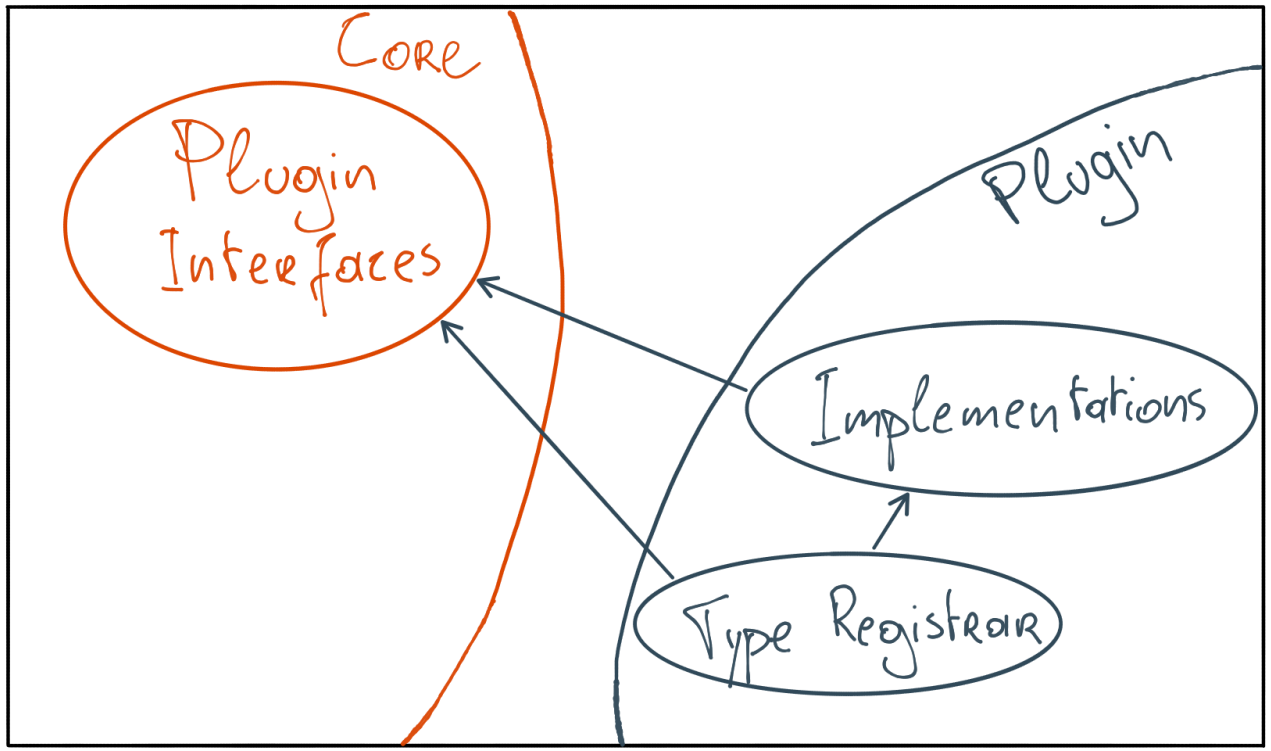
- Usage

```
{
    "$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"
}
```

AXXES

Logic extension points

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



AXXES

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

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

AXXES

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

    public ToyCreator(IToyRepository repository, IScopedServiceLocator 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();

        var creator = _serviceLocator.ResolveGenericType(creatorInterfaceType, toyType);

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

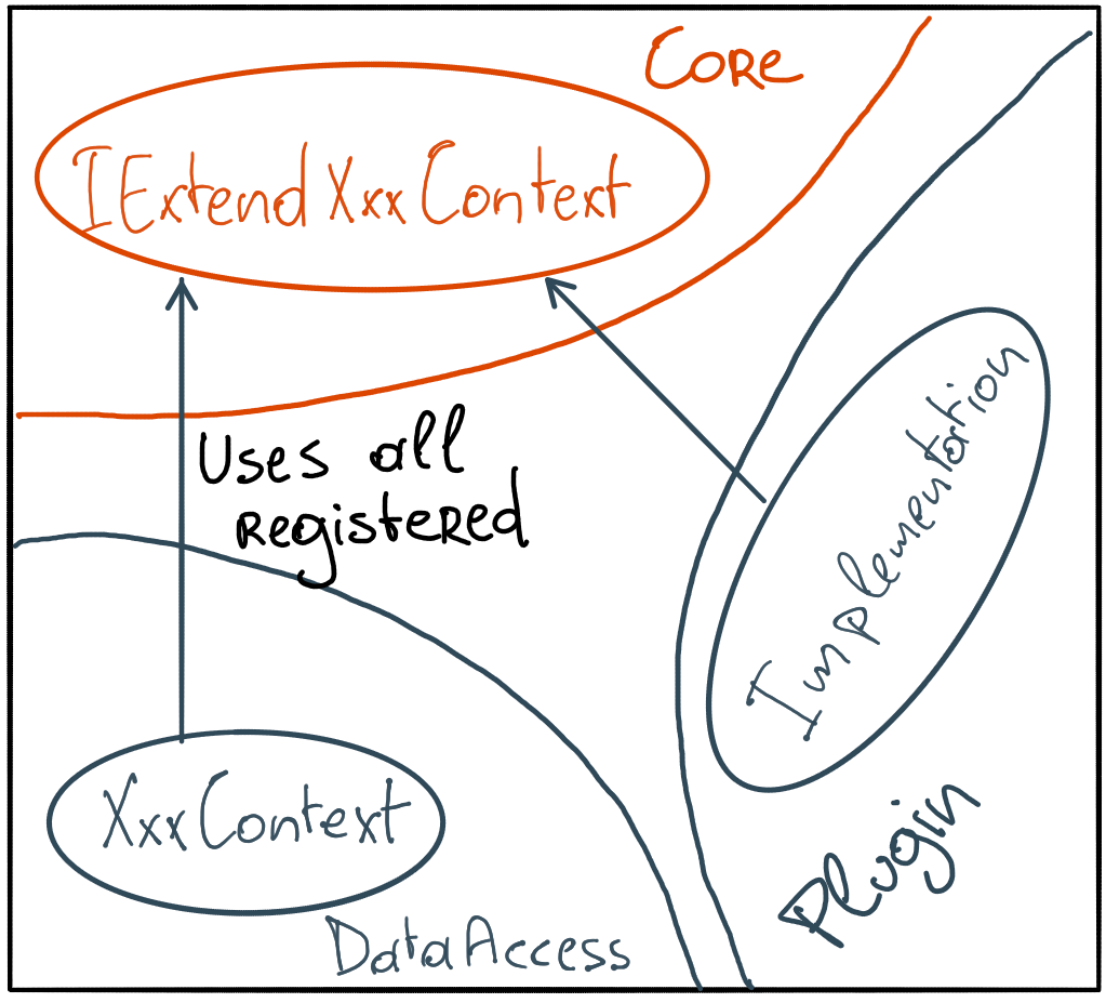
Wrapper that resolves within the current Scope (HttpRequest)

Uses Reflection to construct the generic interface and resolve it

AXXES

Entity Framework

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



AXXES

```
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());
        }
    }
}
```

AXXES

```
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);
        }
    }
```

Database Migrations

Central Approach	Distributed approach
+ Can be generated + Easy to execute at deploy time	+ Every plugin has its own migrations + Database 100% in sync w/ plugins
- All tables/fields exist even if the plugins are not loaded - Will require you to deal with the MigrationHistory table	- Trickier to code & test - Requires runtime migrations

AXXES

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

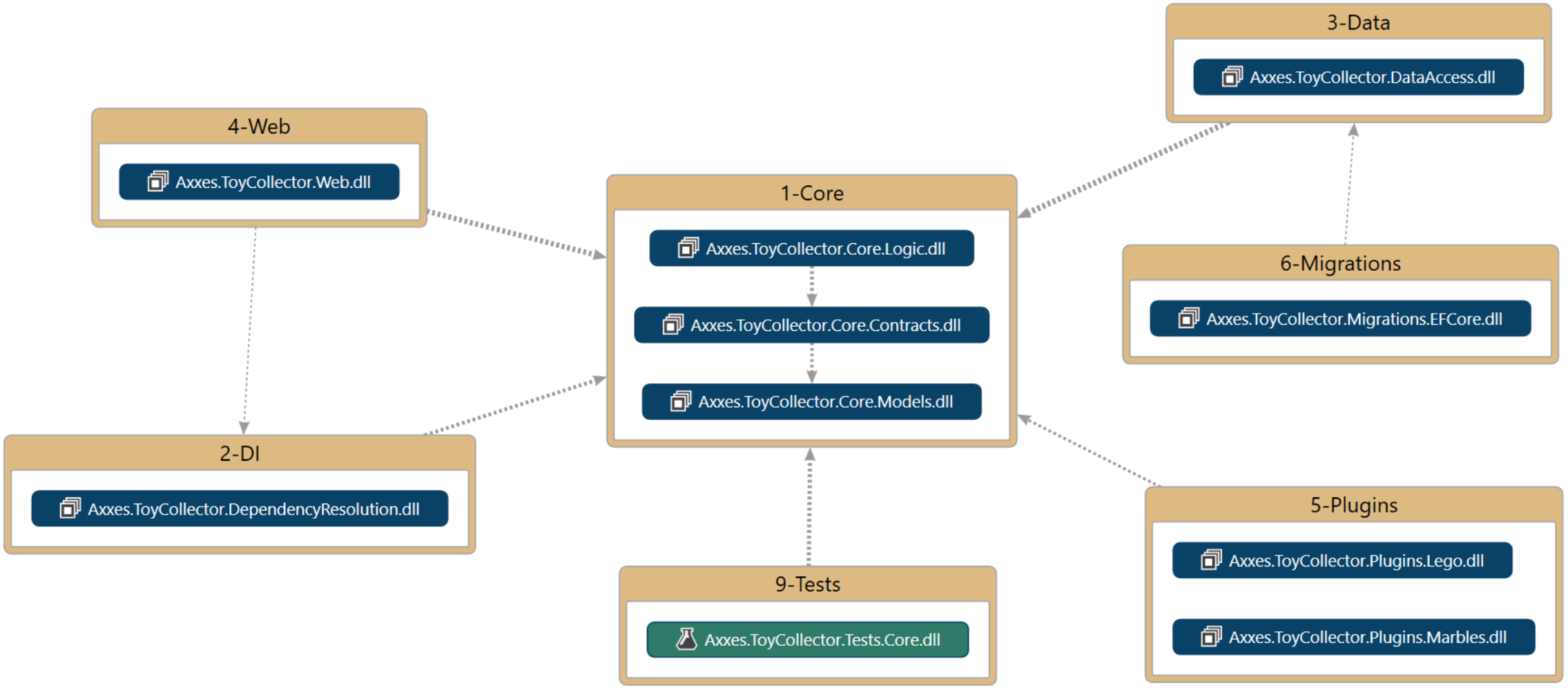
```
[Migration(20181111200901)]
public class CreateToyTable : Migration
{
    public override void Up()
    {
        Create.Table("Toys").InSchema("dbo")
            .WithColumn("Id").AsInt32().PrimaryKey().Identity()
            .WithColumn("Description").AsAnsiString(250).Nullable()
            .WithColumn("Discriminator").AsString(int.MaxValue).NotNullable()
            .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");
    }
}
```

```
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();

        return builder;
    }
}
```

Needs to happen in ONE call

AXXES

Project Dependencies

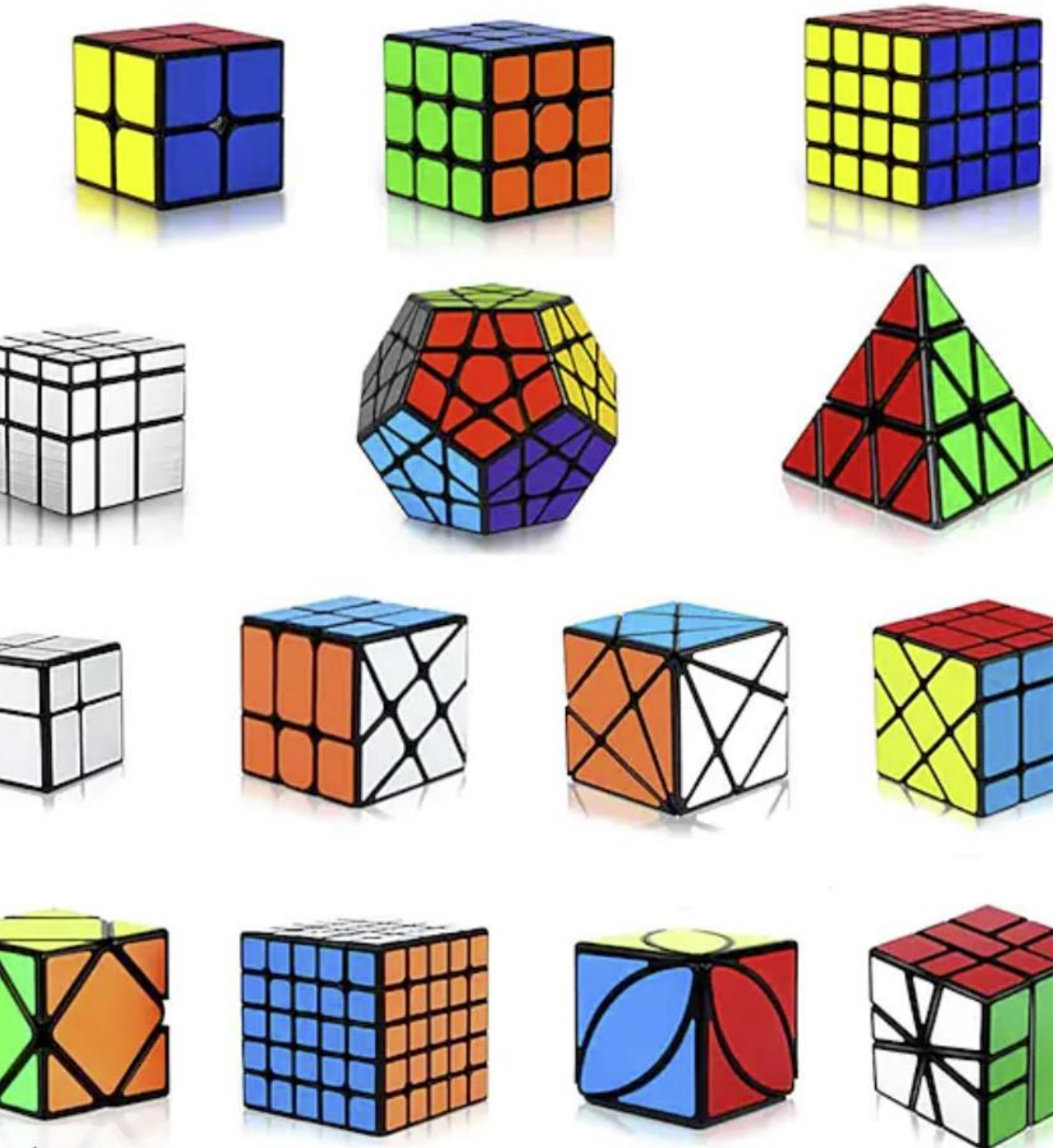


AXXES

Do it yourself!

–

Write an entire plugin yourself!



Cube plugin

1. Make a new assembly for the plugin
2. Make sure it copies to the output of Web
3. Add a controller with a View
4. Add an inherited type Cube (extends Toy)
5. Add some properties to Cube
6. Register Cube in the DbContext
7. Make a migration for Cube & run it
8. Extend the creation of a Cube:
   Write “Ernő Rubik” to the Console.
9. Register the type for the API serializer

… Done!

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.

AXXES

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 …

AXXES

Questions?

About me

Hannes Lowette

Head of Learning & Development at @Axxes_IT

@hannes_lowette

#20086521

Code samples and slides at :

https://github.com/Belenar/Axxes.ToyCollector

AXXES

Thank you!

Entrepotkaai 10A,
2000 Antwerpen

Leonardo Da Vincilaan 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/