Code Reuse in CF

Is Spaghetti Code still Spaghetti if it's DRY?

Online ColdFusion Meetup
May 12, 2022

https://github.com/gpickin/20220509_OnlineCFMLMeetup_CodeReuse

Gavin Pickin

Who am I?

- Software Consultant for Ortus Solutions
- Work with ColdBox, CommandBox, ContentBox every day
- Working with Coldfusion for 22 years
- Working with Javascript just as long
- Love learning and sharing the lessons learned
- From New Zealand, live in Bakersfield, Ca
- Loving wife, lots of kids, and countless critters

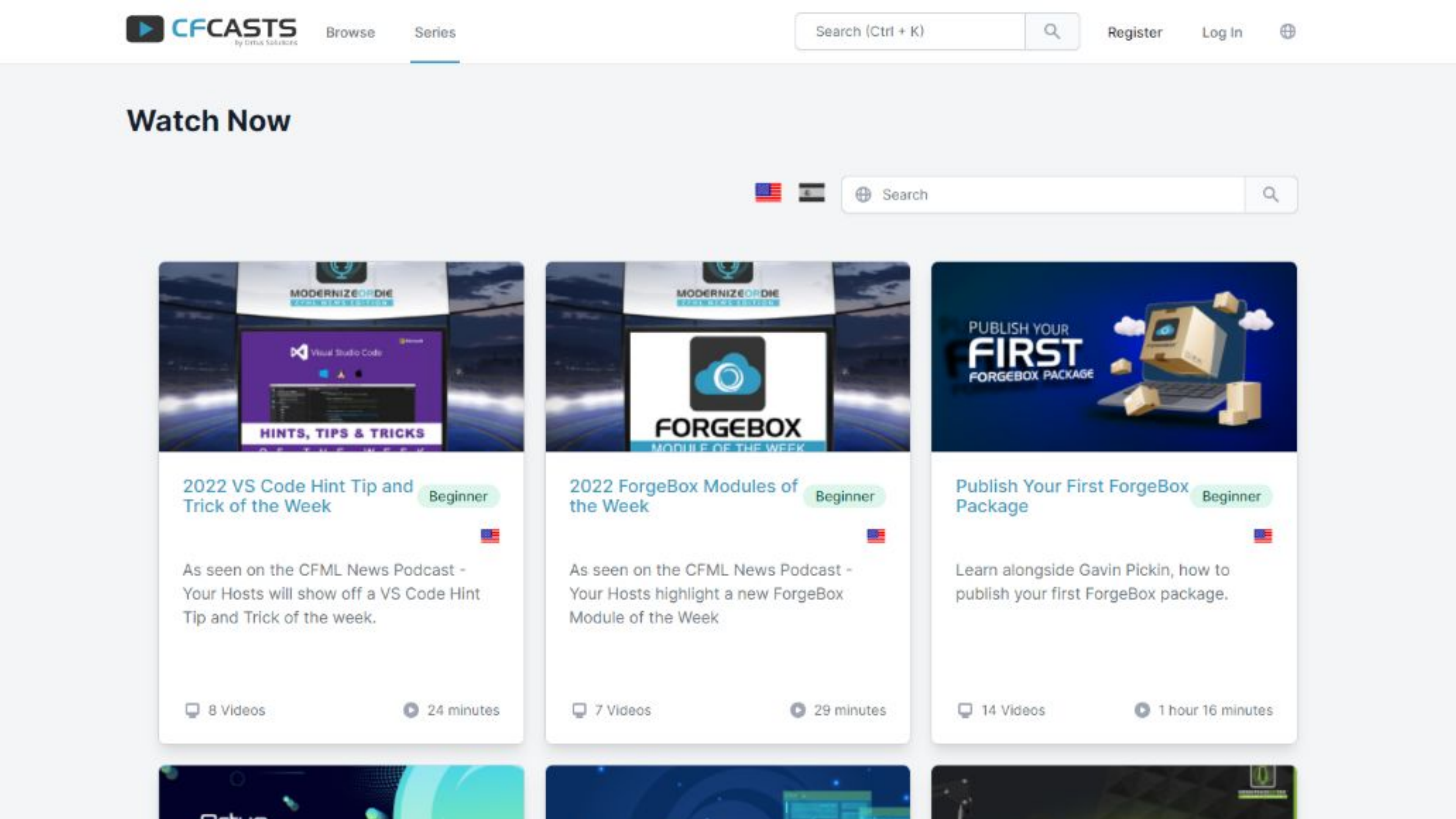
@gpickin on twitter

http://www.ortussolutions.com

Ortus Solutions
A Software Revolution

MODERNIZE OR DIE
CFML NEWS EDITION

Episode 147 - May 10, 2022



CFCASTS by Ortus Solutions   Browse   Series   Search (Ctrl + K)   Register   Log In

Watch Now

Search

2022 VS Code Hint Tip and Trick of the Week Beginner

As seen on the CFML News Podcast - Your Hosts will show off a VS Code Hint Tip and Trick of the week.

8 Videos   24 minutes

2022 ForgeBox Modules of the Week Beginner

As seen on the CFML News Podcast - Your Hosts highlight a new ForgeBox Module of the Week

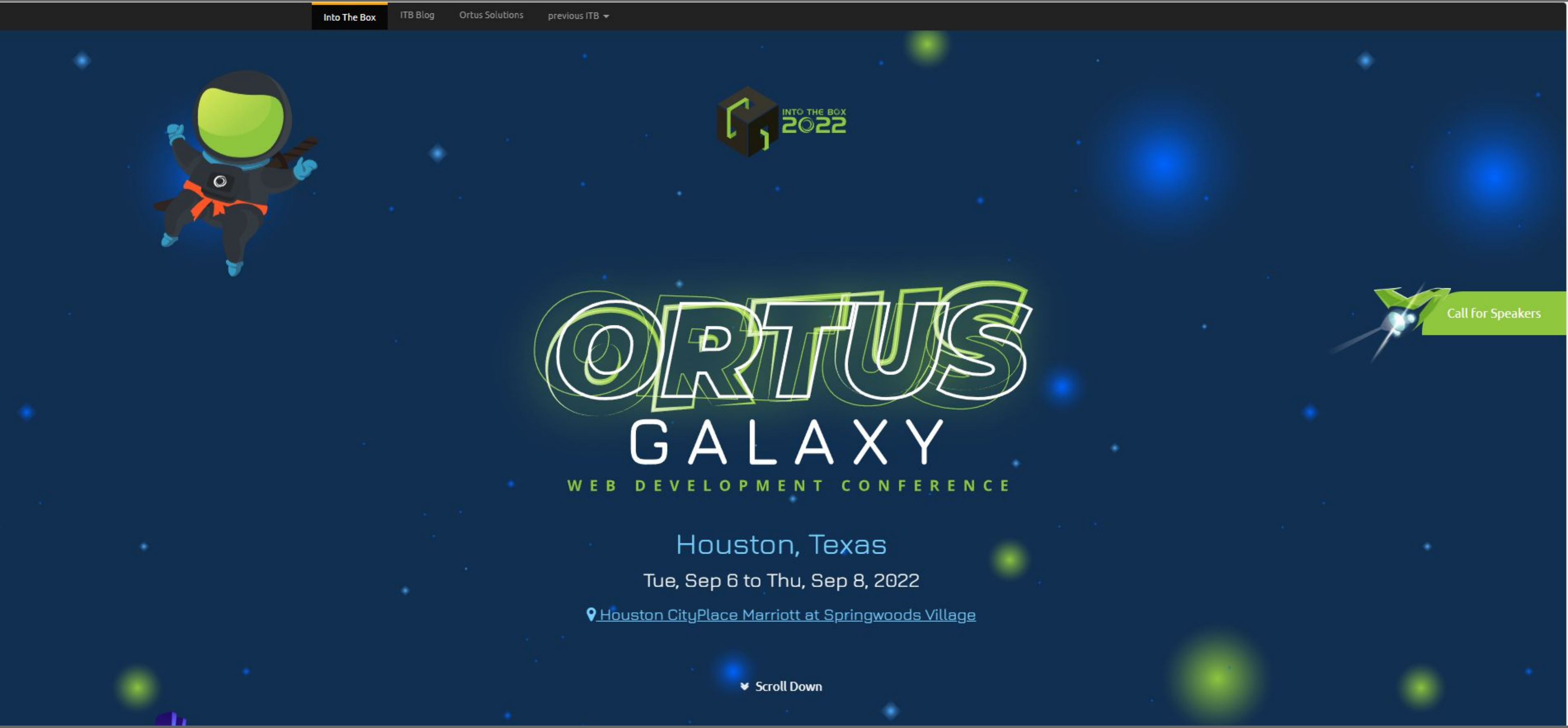
7 Videos   29 minutes

Publish Your First ForgeBox Package Beginner

Learn alongside Gavin Pickin, how to publish your first ForgeBox package.

14 Videos   1 hour 16 minutes

Into the Box 2022 Conference



Houston, Texas

Tue, Sep 6 to Thu, Sep 8, 2022

Houston CityPlace Marriott at Springwoods Village

Into the Box

Early bird prices until May 31st

4 Workshops Announces

2 track Conference

https://intothebox.org/

INTO THE BOX
2022

SPAGHETTI CODE

Spaghetti code usually is formed when different developers (or same developer in different times) add or change code, without a consistent approach.

The result can be unplanned, convoluted, confusing and unmaintainable code.

SPAGHETTI CODE

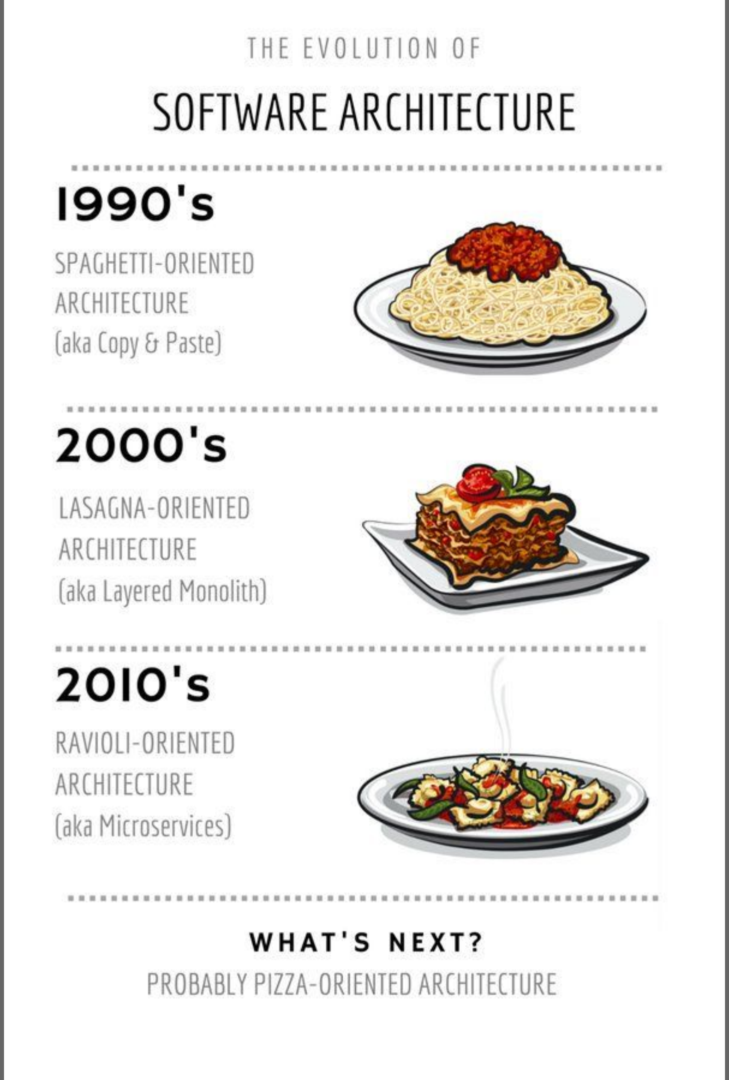
A lot of older sources refer to it as a nest of messy code lacking the structure required to scale effectively.

These are projects where you’re scared to touch it, in case it comes untangles and the world as you know it implodes.

SPAGHETTI CODE

Sometimes it’s a nickname for

LEGACY CODE



THE EVOLUTION OF

SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED ARCHITECTURE (aka Copy & Paste)

2000's

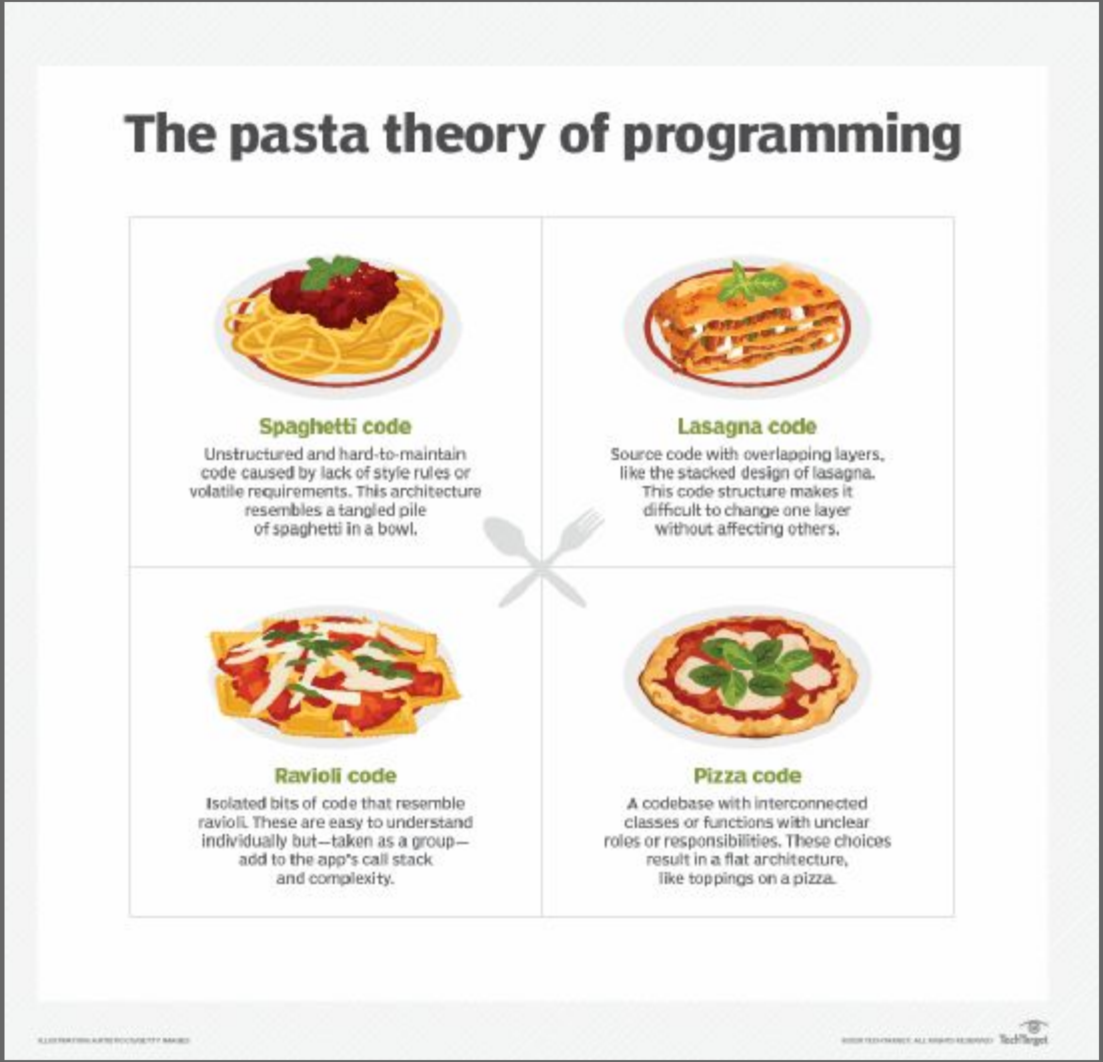
LASAGNA-ORIENTED ARCHITECTURE (aka Layered Monolith)

2010's

RAVIOLI-ORIENTED ARCHITECTURE (aka Microservices)

WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE



The pasta theory of programming

Spaghetti code
Unstructured and hard-to-maintain code caused by lack of style rules or volatile requirements. This architecture resembles a tangled pile of spaghetti in a bowl.

Lasagna code
Source code with overlapping layers, like the stacked design of lasagna. This code structure makes it difficult to change one layer without affecting others.

Ravioli code
Isolated bits of code that resemble ravioli. These are easy to understand individually but—taken as a group—add to the app's call stack and complexity.

Pizza code
A codebase with interconnected classes or functions with unclear roles or responsibilities. These choices result in a flat architecture, like toppings on a pizza.

TechTarget

Related Code Types

Ravioli code:
This is the term for errors in object-oriented code that occur when code is easy to understand in a class but not in the context of the entire project.

Related Code Types

Lasagna code: This is a problem that can occur when you use layers to avoid spaghetti code and the layers are so interdependent on one another that a single break in a layer affects the whole project.

Related Code Types

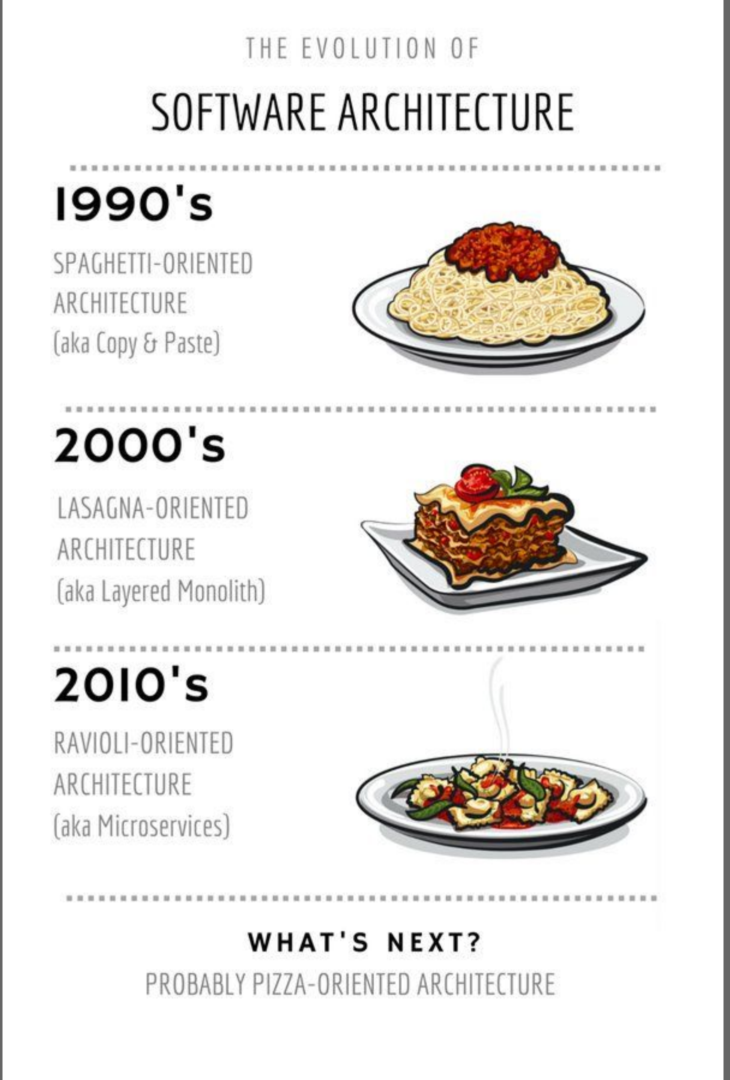
Pizza code: If a code architecture is too flat, it’s called a pizza code.

Source: https://www.bmc.com/blogs/spaghetti-code/

That's not entirely correct. Now we have ravioli stacked up in lasagna-like layers and filled with spaghetti

@mariofusco

https://twitter.com/mariofusco/status/721224776941989888


THE EVOLUTION OF
SOFTWARE ARCHITECTURE
1990's
SPAGHETTI-ORIENTED ARCHITECTURE (aka Copy & Paste)
2000's
LASAGNA-ORIENTED ARCHITECTURE (aka Layered Monolith)
2010's
RAVIOLI-ORIENTED ARCHITECTURE (aka Microservices)
WHAT'S NEXT?
PROBABLY PIZZA-ORIENTED ARCHITECTURE

DRY CODE

Don’t Repeat Yourself (DRY)
The aim to reduce repetition of code.

WET CODE

Write Everything Twice (WET)
Code that doesn’t adhere to DRY principle.

Maintainability

- The biggest benefit of using DRY is maintainability.
- Fixing bugs in one location is easier than remembering and finding all the occurrences of that logic.
- Updating logic or adding functionality is easier in 1 location instead of many

Another Acronym - FIRST

Components Should Be (FIRST):

- Focused
- Independent
- Reusable
- Small &
- Testable

https://addyosmani.com/first/

Readability

- If a developer understands the principles of dry code, they are more likely to understand clean code principles, and that should lead to more readable code.
- Code reuse usually encourages smaller more manageable functions, which promotes lower cognitive load, and therefore more readable.

Programs must be written for people to read, and only incidentally for machines to execute.

Harold Abelson.

Testing

- Usually DRY code is broken down into smaller pieces, usually it is easier to test
- The more code you have, the more paths and functions to cover, the harder to test
- Using DRY Code from libraries and frameworks means more eyes on it, and sometimes those libraries and frameworks have tests for their code already.

Cost

- More code takes more time and money
- Reinventing the wheel takes more time and money
- Maintaining more takes costs more time and money
- Testing more code takes more time and money
- Fixing more bugs takes more time and money

How do we prevent Copy-Pasting?

We need fast and simple ways to share code in a project, and between projects

How to write DRYer code in CFML

Some tools CFML gives you

- CFINCLUDE
- CFMODULE
- Custom Tags
- User Defined Functions (UDFs)
- CFCs with Functions
- Externally Sourced Frameworks / Libraries

CFInclude

- Well known
- Popular
- Easy to use
- Can reference local or absolute locations for easy sharing

Cons:

- It absorbs everything around it
- Hard to control inputs

Lets look at some code

CFInclude

http://127.0.0.1:52820/1_cfinclude/

CFModule

- Black boxes your code, so it only can access what you give it
- Can make your code more flexible, since you can control the inputs
- Can access the caller for access to the parent - can break encapsulation
- Can reference local or absolute locations for easy sharing

Cons

- Not well known
- People don’t seem to like it - especially since it’s equivalent to a custom tag
- Can access the caller for access to the parent - breaks encapsulation and create unintended or unexpected side effects

Lets look at some code

CFModule

http://127.0.0.1:52820/2_cfmodule/

Custom Tags

- Gives you tag syntax to allow you to wrap content nicely (Great for UI)
- Can make your code more flexible, since you can control the inputs
- You can access that tag content inside of your custom tag as well
- Can access the caller for access to the parent - can break encapsulation
- Engines has central Custom Tag locations for sharing between projects

Cons

- Since it wraps code, it executes two times, and confuses people sometimes
- Executes 2 times, even with a self closing tag
- People don’t seem to like it - although it was powerful
- Can access the caller for access to the parent - breaks encapsulation and create unintended or unexpected side effects

Lets look at some code

Custom Tags

http://127.0.0.1:52820/3_customtags/

User Defined Functions (UDFs)

- Can make your code more flexible, since you can control the inputs
- Simple to create, include multiple for simplicity
- Only runs when you call the function
- Can access the calling environment if you need to

Cons

- Can access the calling environment - breaks encapsulation and create unintended or unexpected side effects
- Including lots of functions everywhere can bloat your bytecode?

Lets look at some code

User Defined Functions

http://127.0.0.1:52820/4_udfs/

CFCs with Functions

- Can make your code more flexible, since you can control the inputs
- Simple to create, include multiple in utility cfcs
- Only runs when you call the function
- You can mixin cfm files into CFCs
- You can extend CFCs to other CFCs
- Data doesn’t leak from the page into your function

Cons

- Creating CFCs inside of other CFCs and in every page can bloat your bytecode?
- If you mixin the same functions to every CFC, isn’t that going to bloat your bytecode?

Lets look at some code

CFCs

http://127.0.0.1:52820/5_cfcs/

Importing Tag Libraries

You can use the cfimport tag to import either of the following:

* All CFML pages in a directory, as a tag custom tag library.

* A Java Server Page (JSP) tag library. A JSP tag library is a packaged set of tag handlers that conform to the JSP 1.1 tag extension API.

https://cfdocs.org/cfimport

JVM Functionality

ColdFusion gives you the ability to use CFObject tags to create instances of Java Objects, already available in the JVM that ColdFusion runs on top of.

Method arguments and return values can be any valid Java type, simple values, arrays, and objects.

ColdFusion handles transformation when you pass values into Java Functions and has a series of java interoperability methods for casting values to and from Java.

Common java functionality accessed might include environment variables, java arguments, networking, host, date objects, maps, and much more.

JVM Functionality Cons

- Java is a very verbose language, and sometimes it requires jumping through a lot of extra hoops to do something simple.
- ColdFusion still lacks power features when working with java, like extending a java class or implementing some Interface types.
- Ask Brad Wood; he’s always trying to get those features added to all the ColdFusion Engines.

Tools to help use the Features more effectively and efficiently

- Dependency Injection
- Externally Sourced Frameworks / Libraries
    - CFML
    - Java
- CommandBox Endpoint Installation
- Full ColdFusion Package Management for your App

WireBox and how to manage CFCs

WireBox alleviates the need for custom object factories or manual object creation in your ColdFusion (CFML) applications.

It provides a standardized approach to object construction and assembling that will make your code easier to adapt to changes, easier to test, mock and extend.

Advantages of a DI Framework

Compared to manual Dependency Injection (DI), using WireBox can lead to the following advantages:

- You will write less boilerplate code.
- By giving WireBox DI responsibilities, you will stop creating objects manually or using custom object factories.
- You can leverage object persistence scopes for performance and scalability. Even create time persisted objects.
- You will not have any object creation or wiring code in your application, but have it abstracted via WireBox. Which will lead to more cohesive code that is not plagued with boilerplate code or factory code.
- Objects will become more testable and easier to mock, which in turn can accelerate your development by using a TDD (Test Driven Development), BDD (Behavior Driven Development) approach.
- Once WireBox leverages your objects you can take advantage of AOP or other event life cycle processes to really get funky with OO.

Externally Sourced Frameworks / Libraries

Most importantly, when discussing DRY code, using 3rd party extension frameworks and libraries stops you from reinventing the wheel. Learn from others' mistakes and gain from their security and coding practices. Contributions from many people can give you options you haven’t thought about yet.

CommandBox - Endpoint Installation

For CommandBox to install packages for you, it needs to connect to where packages are stored so it can download them for installation. CommandBox integrates seamlessly with ForgeBox, our community of ColdFusion (CFML) projects, which is the recommended Endpoint. CommandBox also integrates with many other endpoints to use with CFML and Java.

CommandBox - Endpoint Installation

- HTTP(S) - Point to a hosted zip file containing a package
- File - A local file containing a package
- Folder - A local folder containing a package
- Git - Any Git repo containing a package
- Lex - A Lucee Extension hosted via HTTP that's *not* contained in a zip file
- S3 - A package zip stored in a private S3 bucket

CommandBox - Endpoint Installation

- CFLib - UDFs posted on [CFLib.org](http://cflib.org/)
- Gist - A package hosted as a Gist from gist.github.com
- Java - Install OpenJDK for your servers
- Jar - A jar file hosted via HTTP that's *not* contained in a zip file

Full ColdFusion Package Management for your App

ForgeBox is the only full package management solution for your ColdFusion Applications, accessible from the CLI with CommandBox, and accessible by the browser at ForgeBox.io.

TLDR or TLDL or TLDR

ColdFusion gives you a lot of great features and ways to leverage Code Reuse and there are lots of great tools to help you effectively, efficiently, and easily reuse your code.

Can code be too dry?

- Moderation is the key to most things
- You don’t want overdry cake, same with your code

- Don’t prematurely optimize your Code
- Don’t assume 2 pieces of code should be combined, sometimes the differences warrant code separation.

Is Dry Spaghetti still Spaghetti?

- If you DRY your Code, it doesn’t run blood red with all your suffering.
- You might have some raviolis in there, some lasagna, and maybe some pizza.
- At least those are more bite size, less messy, less tangly, easier to consume

Your code is still Italian, it’s a little strange and don’t under everything it says, but you’re on your way!!!

Who’s hungry???

Questions

???

Thanks

Join me on June 9 for my next Online CF Meetup

Why should I use 3rd party libraries vs Roll my own

- https://www.ortussolutions.com/
- https://intothebox.org/
- https://cfcasts.com/
- https://www.youtube.com/ortussolutions
- https://cfmlnews.modernizeordie.io/