



Design Principles For Microservices



Me?

- * Almost 20 years experience
- * Polyglot programmer, trainer, and coach at Mozaic Works
- * Author
- * Software Crafter
- * Speaker and facilitator around Europe






Recently, YouTuber

THINK. DESIGN. WORK SMART.

Mozaic Works



Think. Design. Work Smart.

SUBSCRIBED

HOMEVIDEOSPLAYLISTSCHANNELSDISCUSSIONABOUT

Think. Design. Work smart.


0 1:17

Mozaic Works Channel


Think. Design. Work Smart. • 71 views • 2 months ago

If you are looking for inspiration and a new source of practical knowledge, we are welcoming you to the Mozaic Works video channel on YouTube: Think. Design. Work Smart. Practical experience shar...


FEATURED CHANNELS

 CppEurope

SUBSCRIBE

 I.T.A.K.E. Unconference




SUBSCRIBE

 Adrian Bolboaca

SUBSCRIBE

Uploads ▶ PLAY ALL

Agile CHALLENGES






Project Management

9:49

Peter Hilton - Advice for Technical Documentation

3 views • 2 days ago

C++ CHALLENGES






Cross Platform Microsoft Tools

1:00:03

Elizabeth Morrow & Augustin Popa - C++ cross-platform...

1 view • 5 days ago

Agile CHALLENGES





Design Patterns, Architecture and the PLoP...

1:12:40

Joe Yoder - Design Patterns, Architecture and the PLoP...

4 views • 1 week ago

Remote Work in Agile





Remote Work in Agile

44:57

Remote Work in Agile

10 views • 2 weeks ago

Agile CHALLENGES



Big Ball of Mud with Alex Bolboaca

3:27

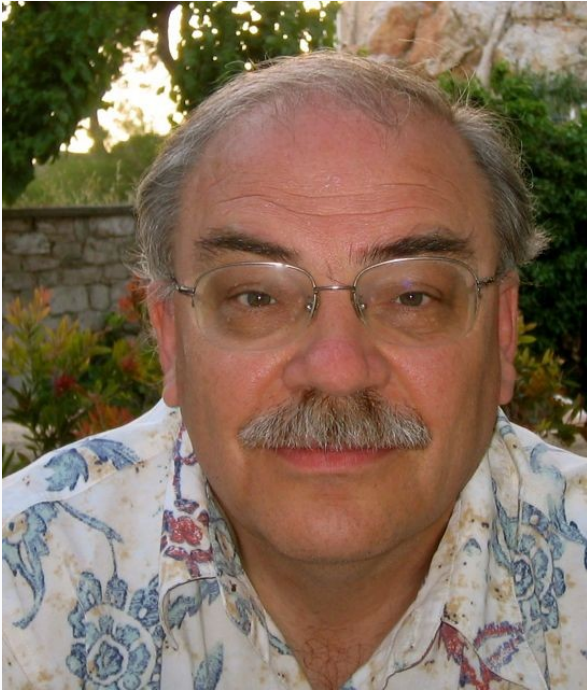
Joe Yoder - Excerpt: Big Ball of Mud with Alex Bolboaca

7 views • 2 weeks ago





Why was OOP necessary?



Allen Wirfs-Brock,
pioneer of personal computing

- * Structuring the code for simulations
- * Some simulations are based on maths
- * Others require messaging



Even weirder connections



Alan Kay

- * “Every object should have an URL”
- * OOP meant “messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things”



The similarity is uncanny



Photo by [Holger Link](#) on [Unsplash](#)



So, design principles

- * UNIX design principles
- * Low coupling, high cohesion
- * SOLID Principles



UNIX Design Principles

- * **Rule of Modularity:** Write simple parts connected by clean interfaces.
- * **Rule of Composition:** Design programs to be connected to other programs.
- * **Rule of Separation:** Separate policy from mechanism; separate interfaces from engines.
- * **Rule of Parsimony:** Write a big program only when it is clear by demonstration that nothing else will do.
- * **Rule of Representation:** Fold knowledge into data so program logic can be stupid and robust.
- * **Rule of Extensibility:** Design for the future, because it will be here sooner than you think.



UNIX Design Principles

*“This is the Unix philosophy:
Write programs that do one thing and do it well.
Write programs to work together.
Write programs to handle text streams,
because that is a universal interface.”*



The similarity is uncanny

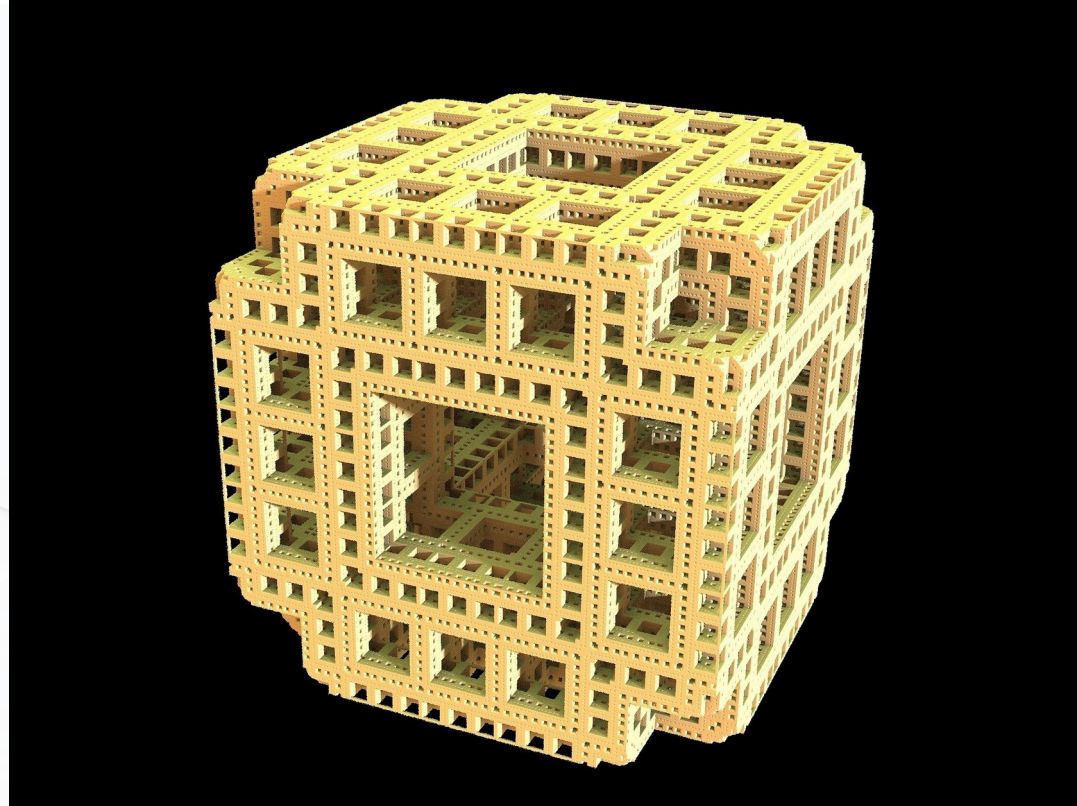


Photo by [Holger Link](#) on [Unsplash](#)



Perhaps old principles apply

- * **Microservices are a new iteration of older ideas:**
 - modularity
 - managing complexity
 - parallel development
- * **But microservices moved at a higher level**



Encapsulation



- * A language mechanism for restricting direct access to some of the object's components.
- * A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.



Encapsulation for Microservices

- * Each microservice with its own database
- * Nobody else can access a microservice database
- * Currently hidden under a kind of web API



Low Coupling



- * **Coupling** = the degree of interdependence between software modules
- * Low coupling in software = reduced coupling surface



Low coupling for microservices?

- * Lack of dependencies between the components
- * Lack of knowledge of another microservice
- * Indirection through an event bus



High Cohesion



- * Cohesion = “the degree to which the elements inside a module belong together”



High cohesion for microservices?

- * Small
- * Interface as small as possible, but not smaller



Single Responsibility Principle

- * “every **module** or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class, **module** or function”



Open Closed Principle

- * “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”
- * that is, such an entity can allow its behaviour to be extended without modifying its source code.



So, applying design principles

- * Low coupling, high cohesion, SRP, ISP, OCP → Small microservices, with a single goal
- * Encapsulation, high cohesion, OCP → own database
- * Low coupling → communicate through events



What about LSP?

- * if S is a subtype of T , then objects of type T may be replaced with objects of type S (i.e. an object of type T may be substituted with any object of a subtype S) without altering any of the desirable properties of the program



Modularity





Modularity

“the degree to which a system's components may be separated and recombined, often with the benefit of flexibility and variety in use”



Replaceable Modules?

- * LSP is about replaceability in classes
- * But classes are classifications of objects
- * “Classes” of microservices need to share the API and the contract to be replaceable



In an image...



