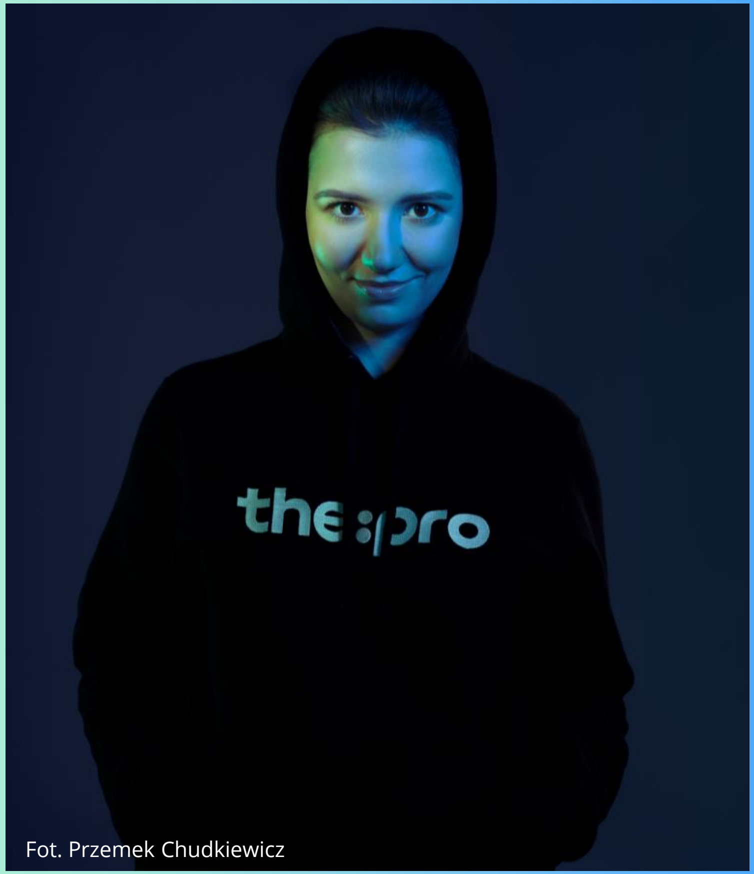
Zdrowe związki poradnik programisty

cohesion & coupling

the:protocol

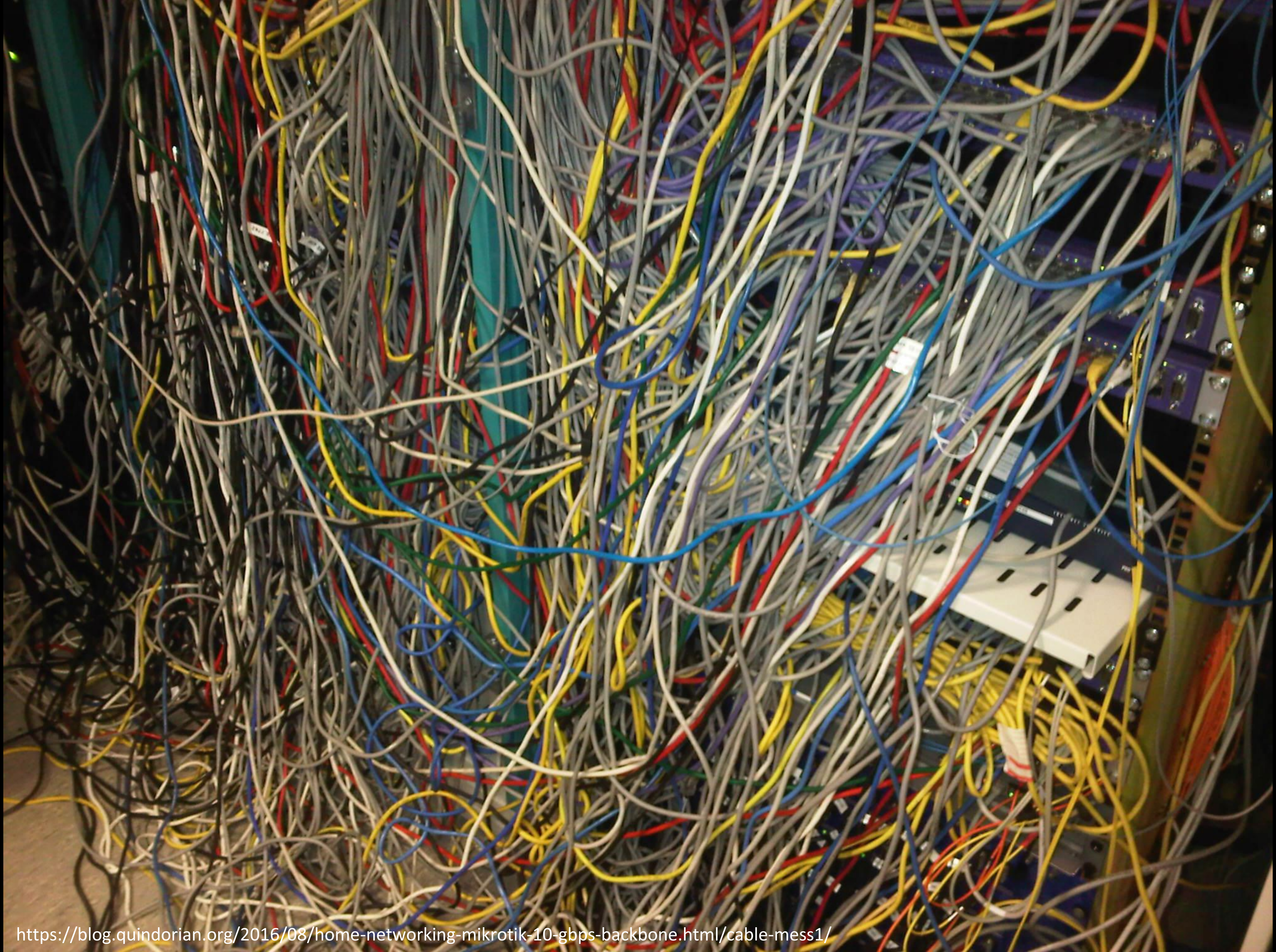


Fot. Przemek Chudkiewicz

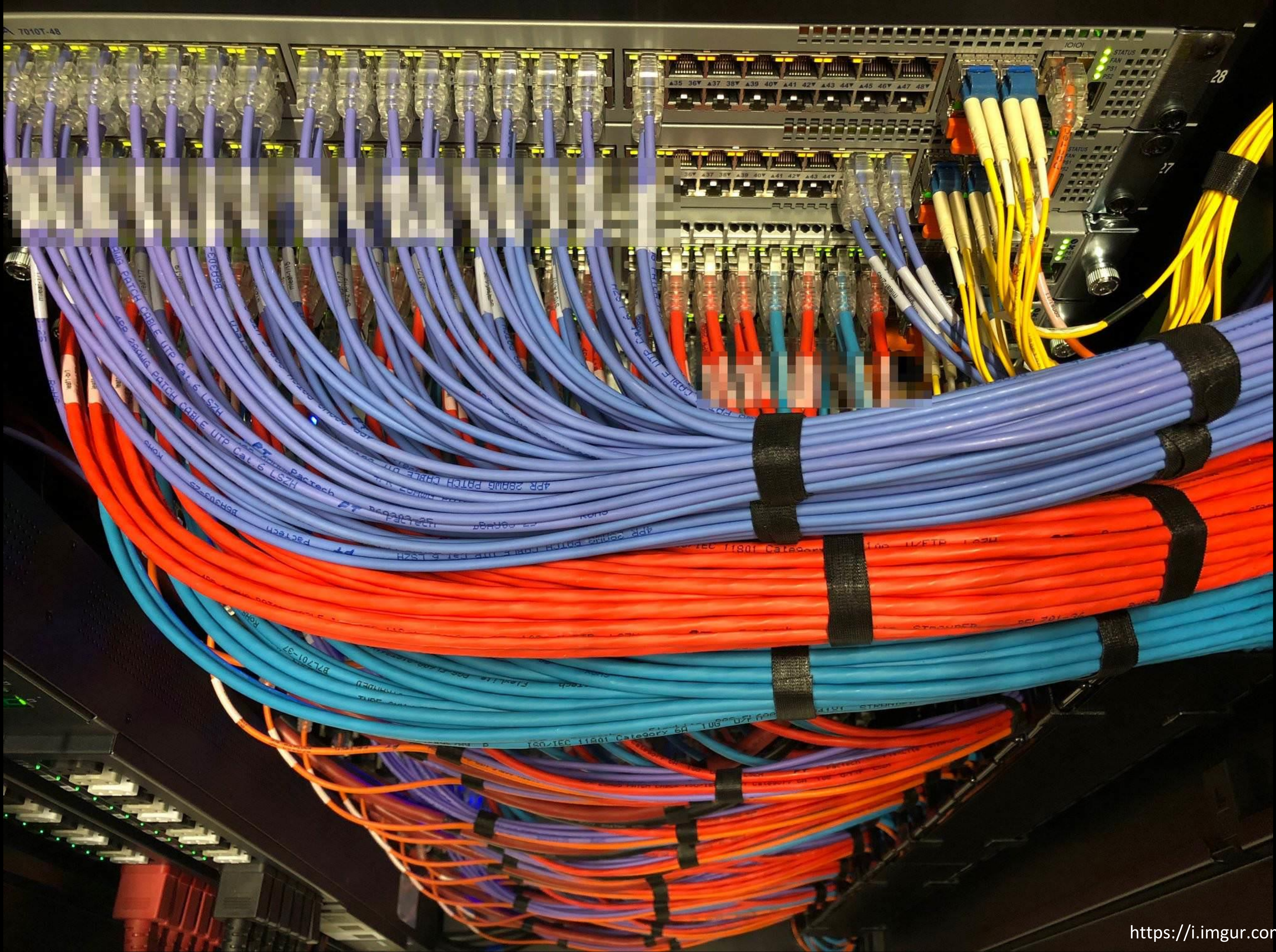
kinga.gazdzinska@pracuj.pl
architektka oprogramowania
grupa pracuj

the:protocol

good design



https://blog.quindorian.org/2016/08/home-networking-mikrotik-10-gbps-backbone.html/cable-mess1/



https://i.imgur.com/5EHec1w.jpg

„Programmers can add features steadily to well-designed software.”

-Kent Beck

Kent Beck
@KentBeck

Programmer, coach coach, singer/guitarist, peripatetic. Learning to be me. Works at @GustoHQ.

661 Following 162.9K Followers

the:protocol

goal of software design: **cost** minimization

↑

cost of **maintenance**

↑

cost of changes that **ripple through the system**

effective software design → minimize chance of change propagation

Yourdon and Constantine in Structured Design

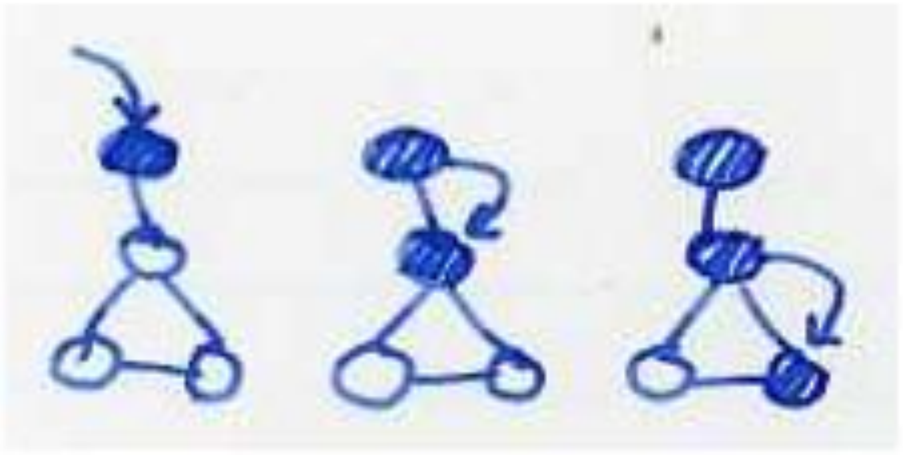
„In particular, **coupling** and **cohesion** play a central role in the value of software design.”

-Kent Beck

coupling

two elements are coupled to the degree that changes to one tend to require changes in another

coupling between elements propagates change



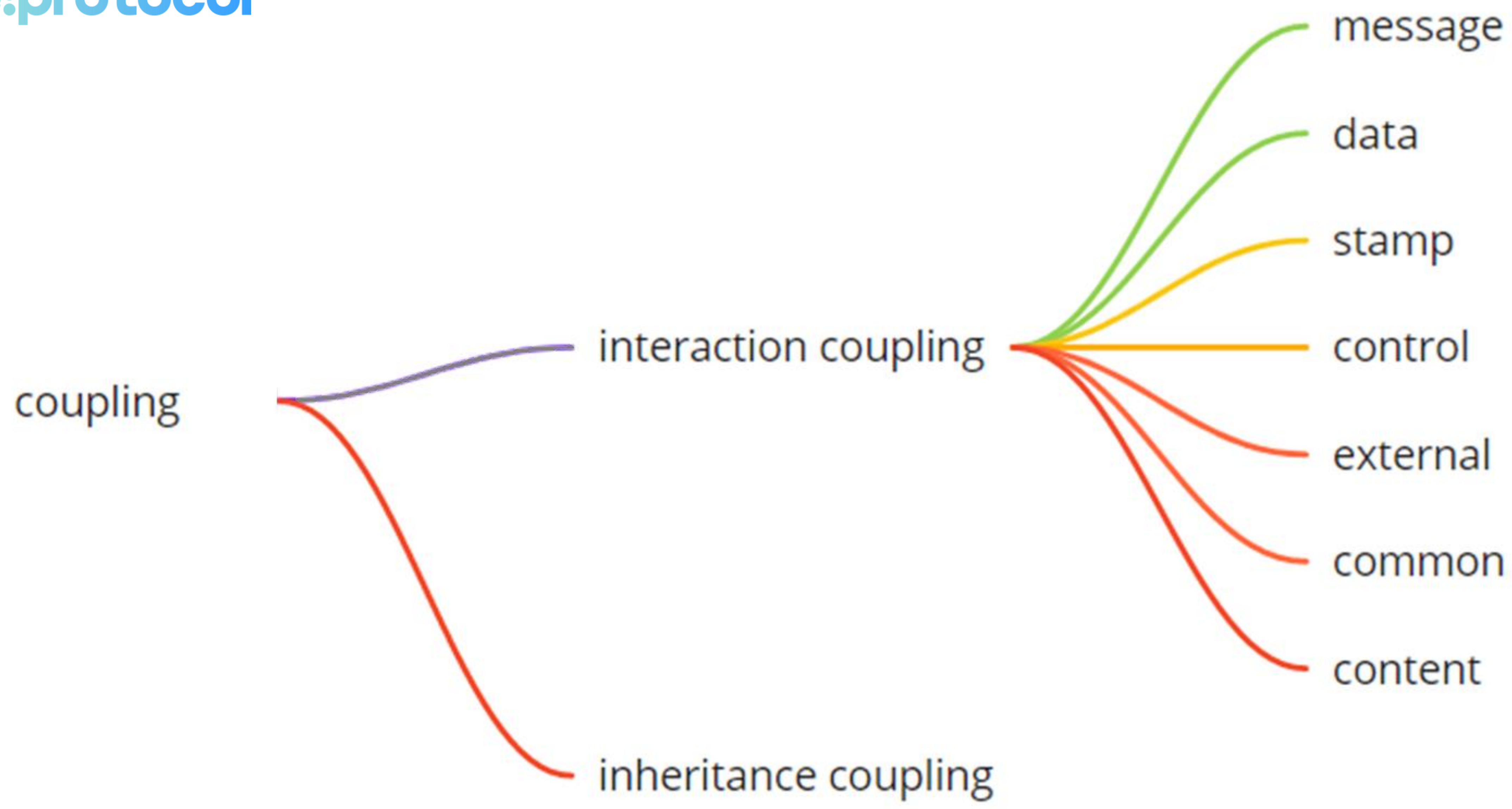
If there is no change, the coupling doesn’t matter.

Elements in a design should not be coupled with respect to the changes that actually take place. This keeps the cost of a change contained.

Changes that are likely to be expensive are less likely to be chosen.

Breaking a coupling can open up new possibilities.

the:protocol



is coupling too high?

- Divergent Change
- Feature Envy
- Inappropriate Intimacy
- Message Chains
- Middle Man
- Shotgun Surgery

Law of Demeter (Principle of Least Knowledge)

“talk only to components directly close to you”

> Method M on Object O can only invoke methods of the following objects:
>
> 1. On O itself.
> 2. On objects contained in attributes of itself (O) or a superclass (of O).
> 3. On an object that is passed as a parameter to the Method M.
> 4. On an object that is created by the Method M.

the:protocol

good coupling:

codebase clean

dependencies easy to understand

bad coupling:

codebase – entangled mess

dependencies hard to understand

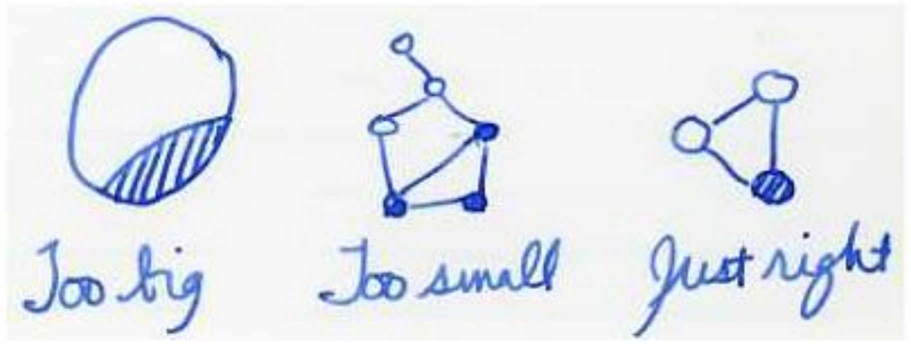
cohesion

measures the cost of a change within an element

element is cohesive to the degree that the entire element changes when the system needs to change

lack of cohesion:

- element too large
- element too small



attempting to divide a cohesive module -> increased coupling & decreased readability

adhesion vs cohesion

root word "hesion" = stick

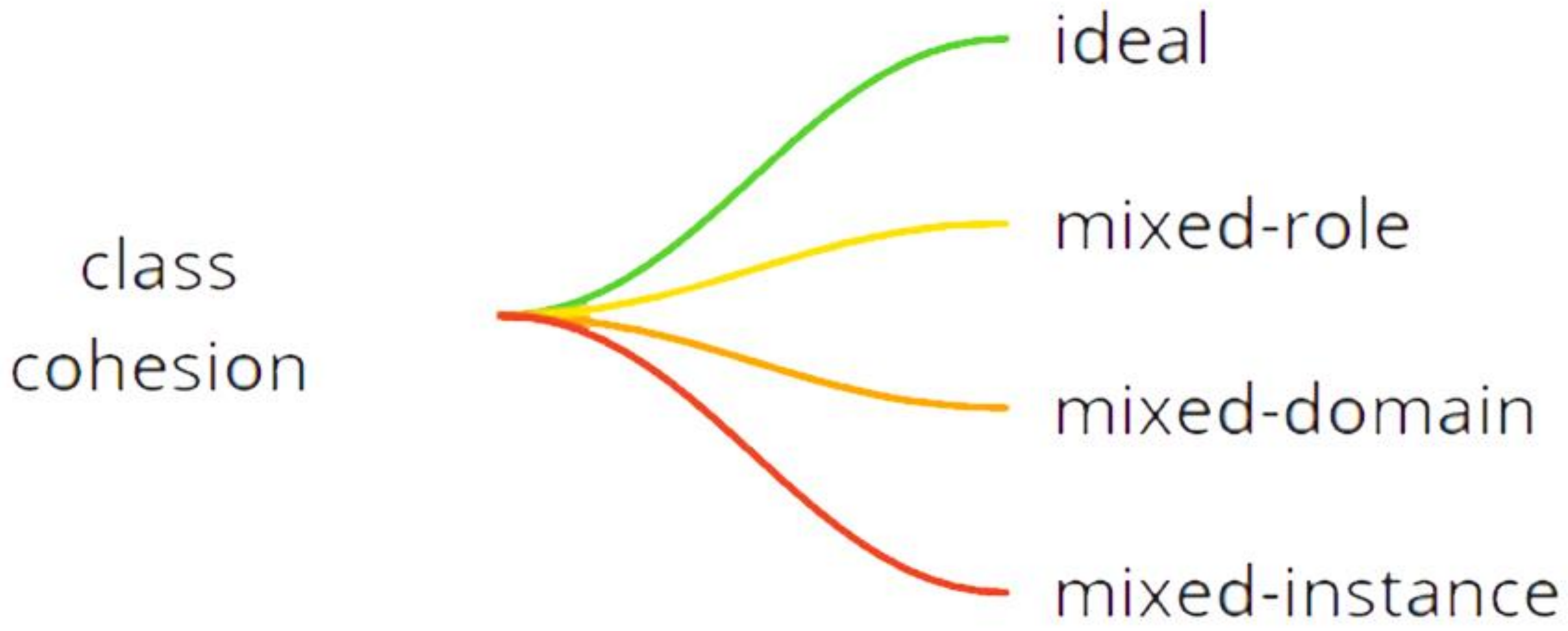
adhesion – clinging of **unlike** molecules

cohesion – clinging of **like** molecules

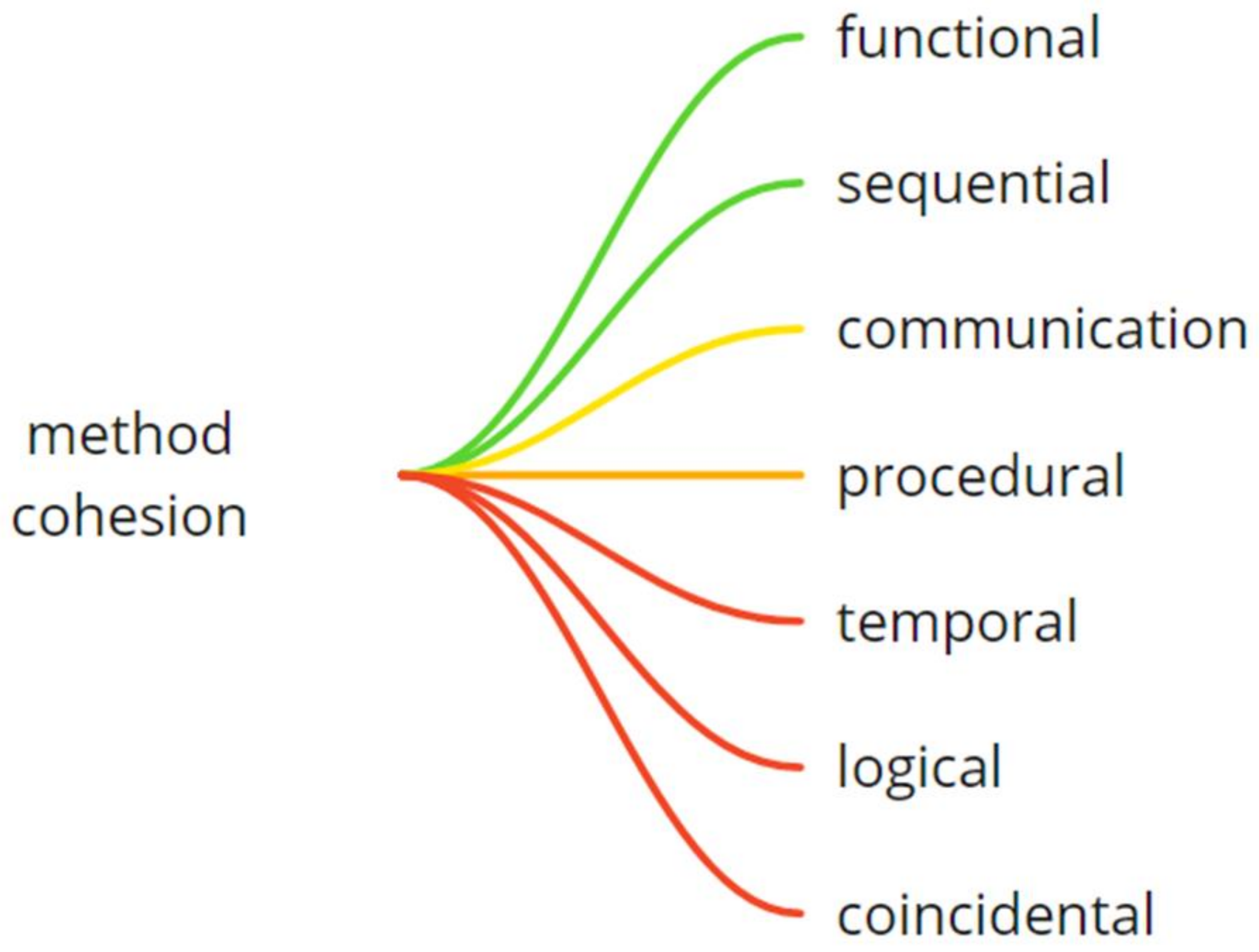
https://www.softschools.com/difference/adhesion_vs_cohesion/38/

https://www.flickr.com/photos/10721976@N08/947643602/

the:protocol



the:protocol



is cohesion too low?

- Data Class
- Lazy Class
- Middle Man
- Primitive Obsession
- Shotgun Surgery

- Static methods in a class
- Helper classes
- Criteria for grouping things (for example, in buckets with names like helper, core, utilities, etc.)

is cohesion too high?

- Data Clumps
- Divergent Change
- Duplication
- Large Class
- Long Method
- Long Parameter List

good cohesion:

- component has few responsibilities
- component has simple interface

bad cohesion:

- component has many responsibilities
- component has complex interface

„Code should actually represent the idea it names”

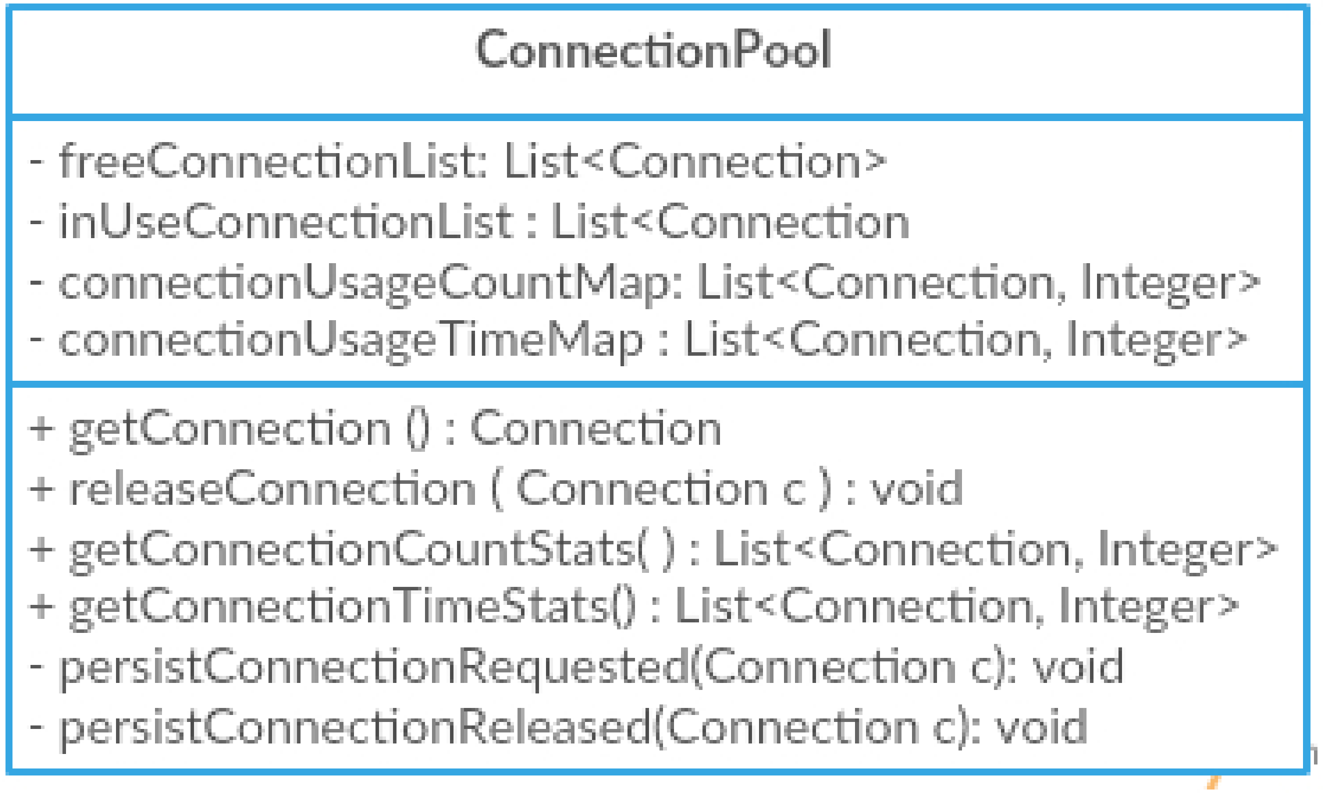
„The worst cases of cohesion are like leaving windows broken on a building – they invite others to break the remaining windows because no one seems to care.”

„Talk is cheap.

Show me the code.”

-Linus Torvalds

With **low cohesion** we could design a `ConnectionPool` class by forcefully stuffing all this functionality/responsibilities into a single class as below. We can see that this single class is responsible for connection management, interacting with database as well maintaining connection stats.

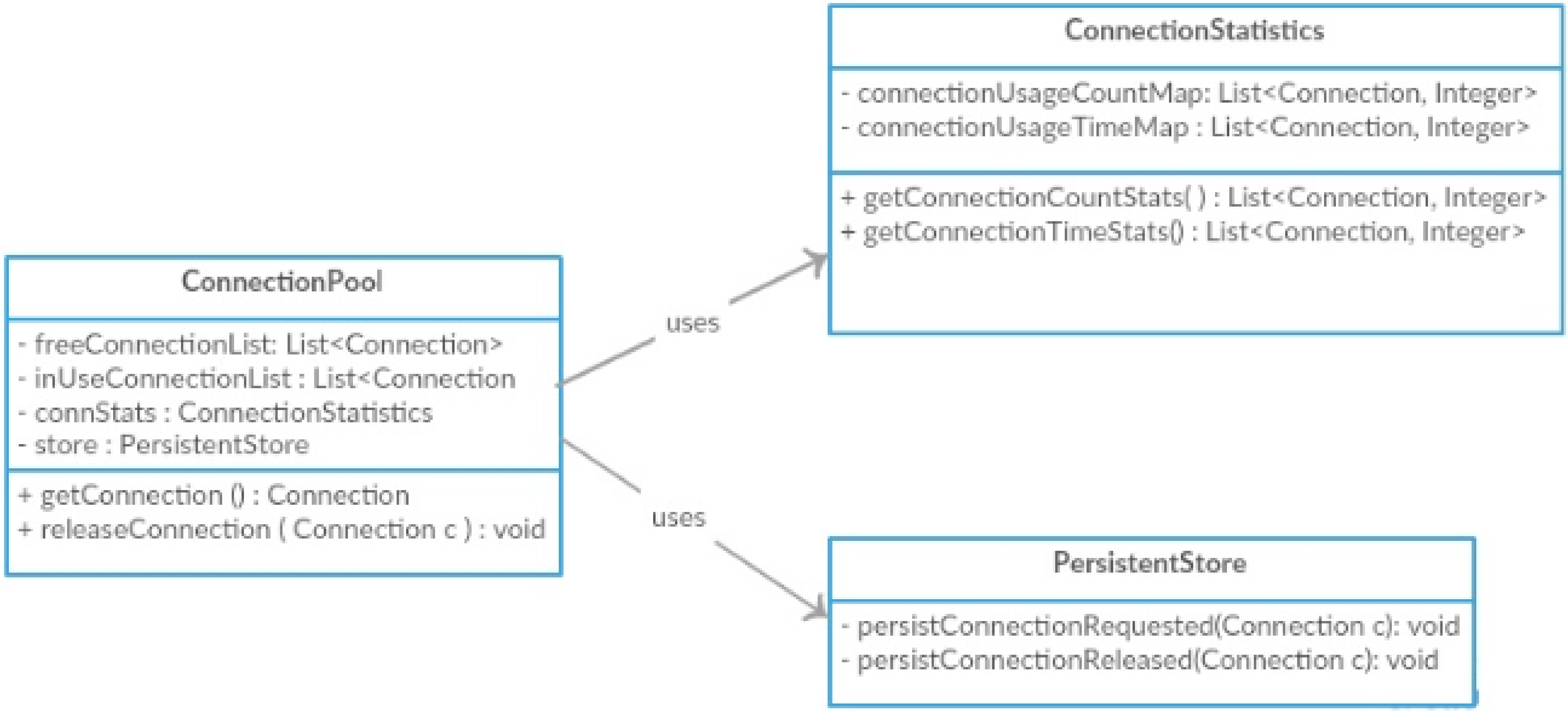

ConnectionPool

- freeConnectionList: List<Connection>
- inUseConnectionList : List<Connection
- connectionUsageCountMap: List<Connection, Integer>
- connectionUsageTimeMap : List<Connection, Integer>

+ getConnection () : Connection
+ releaseConnection (Connection c) : void
+ getConnectionCountStats( ) : List<Connection, Integer>
+ getConnectionTimeStats() : List<Connection, Integer>
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void

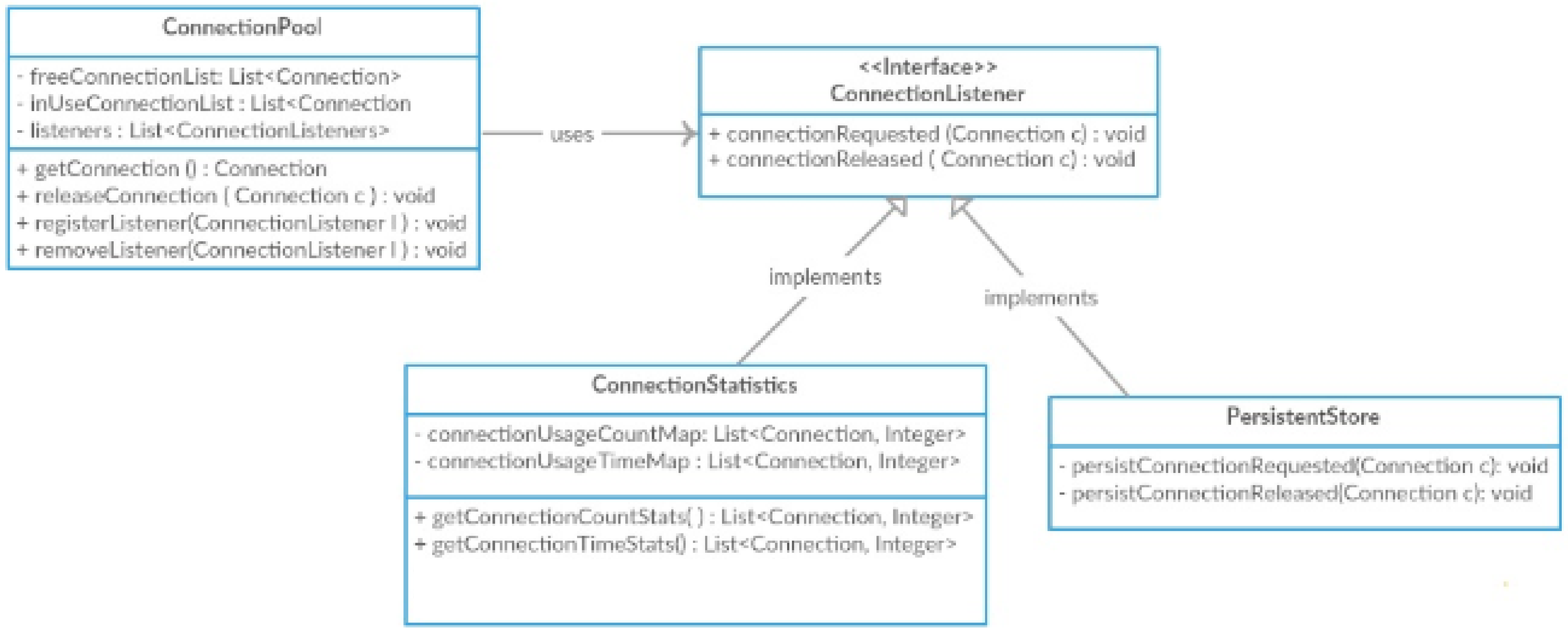
https://stackoverflow.com/a/34634568/12490367

With **high cohesion** we can assign these responsibility across the classes and make it more maintainable and reusable.


ConnectionStatistics
- connectionUsageCountMap: List<Connection, Integer>
- connectionUsageTimeMap : List<Connection, Integer>
+ getConnectionCountStats( ) : List<Connection, Integer>
+ getConnectionTimeStats() : List<Connection, Integer>
ConnectionPool
- freeConnectionList: List<Connection>
- inUseConnectionList : List<Connection
- connStats : ConnectionStatistics
- store : PersistentStore
+ getConnection () : Connection
+ releaseConnection (Connection c) : void
uses
uses
PersistentStore
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void

https://stackoverflow.com/a/34634568/12490367

To demonstrate **Low coupling** we will continue with the high cohesion `ConnectionPool` diagram above. If we look at the above diagram although it supports high cohesion, the `ConnectionPool` is tightly coupled with `ConnectionStatistics` class and `PersistentStore` it interacts with them directly. Instead to reduce the coupling we could introduce a `ConnectionListener` interface and let these two classes implement the interface and let them register with `ConnectionPool` class. And the `ConnectionPool` will iterate through these listeners and notify them of connection get and release events and allows less coupling.

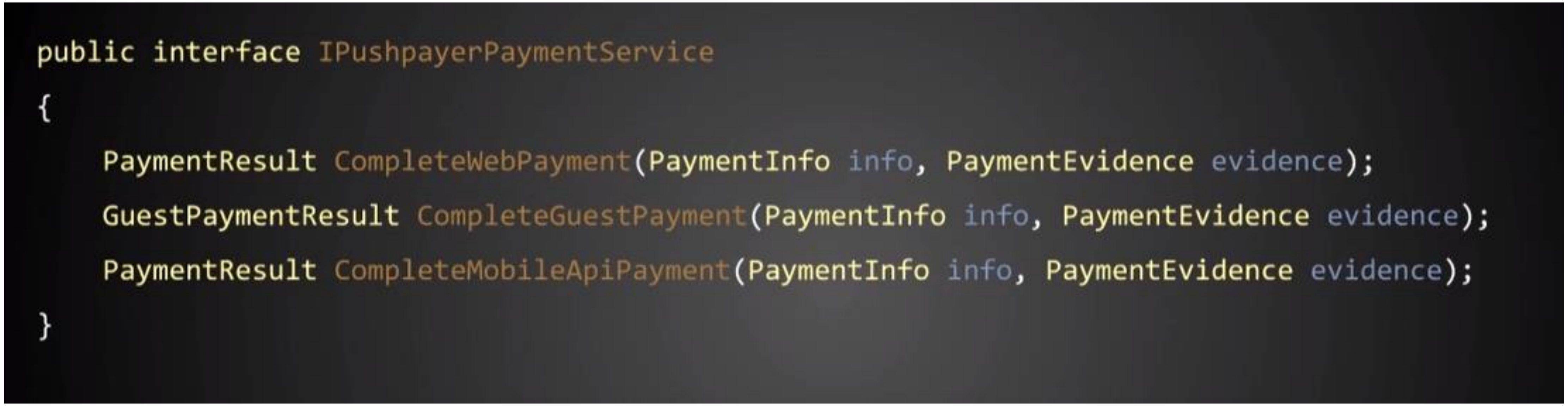

ConnectionPool
- freeConnectionList: List<Connection>
- inUseConnectionList : List<Connection
- listeners : List<ConnectionListeners>
+ getConnection () : Connection
+ releaseConnection (Connection c) : void
+ registerListener(ConnectionListener l) : void
+ removeListener(ConnectionListener l) : void
uses
<<Interface>>
ConnectionListener
+ connectionRequested (Connection c) : void
+ connectionReleased (Connection c) : void
implements
implements
ConnectionStatistics
- connectionUsageCountMap: List<Connection, Integer>
- connectionUsageTimeMap : List<Connection, Integer>
+ getConnectionCountStats( ) : List<Connection, Integer>
+ getConnectionTimeStats() : List<Connection, Integer>
PersistentStore
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void

https://stackoverflow.com/a/34634568/12490367

Adhesive Code

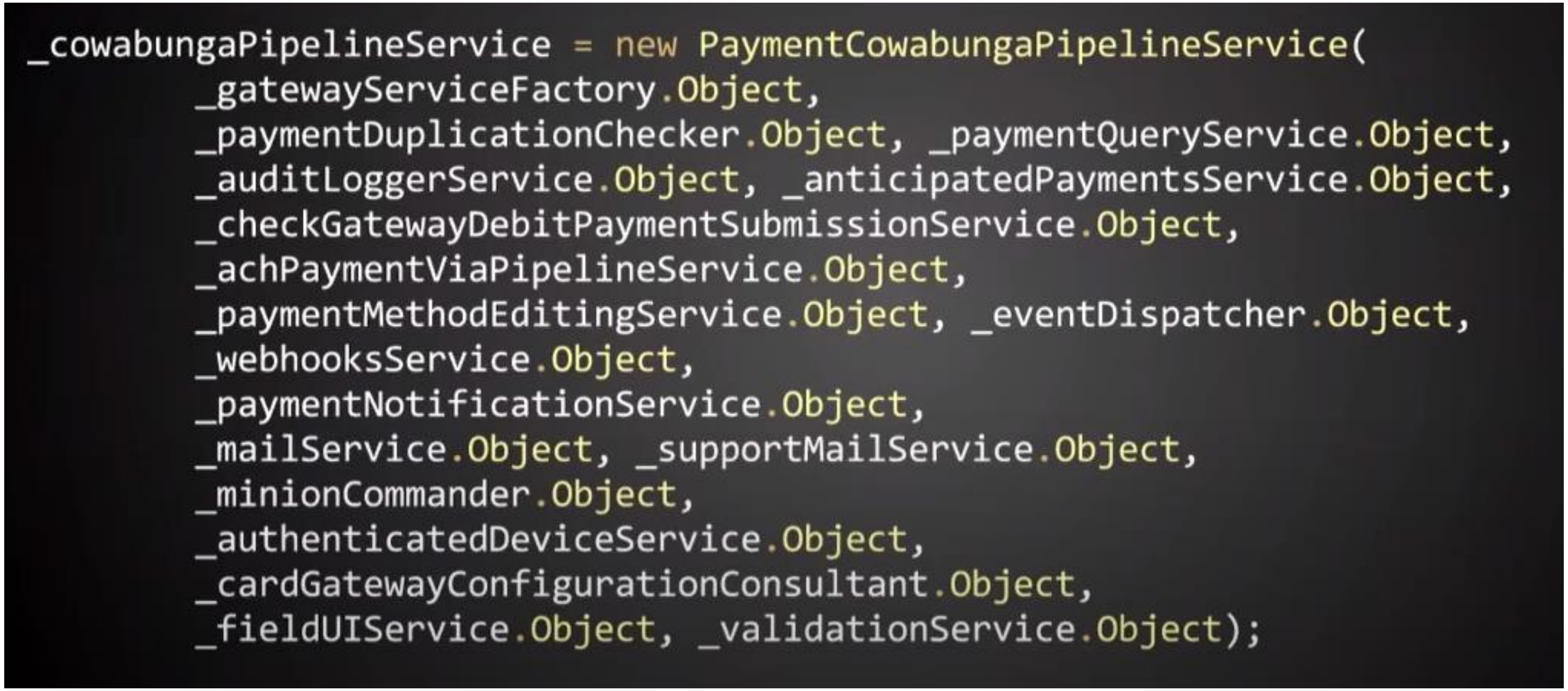
```
public interface IMonolithicDataService
{
    … dozens of methods dealing with
    … tons of different concerns
}
```

cohesion ✓ coupling ?



```
public interface IPushpayerPaymentService
{
    PaymentResult CompleteWebPayment(PaymentInfo info, PaymentEvidence evidence);
    GuestPaymentResult CompleteGuestPayment(PaymentInfo info, PaymentEvidence evidence);
    PaymentResult CompleteMobileApiPayment(PaymentInfo info, PaymentEvidence evidence);
}
```

test mocks setup exposes coupling



```
_cowabungaPipelineService = new PaymentCowabungaPipelineService(
        _gatewayServiceFactory.Object,
        _paymentDuplicationChecker.Object, _paymentQueryService.Object,
        _auditLoggerService.Object, _anticipatedPaymentsService.Object,
        _checkGatewayDebitPaymentSubmissionService.Object,
        _achPaymentViaPipelineService.Object,
        _paymentMethodEditingService.Object, _eventDispatcher.Object,
        _webhooksService.Object,
        _paymentNotificationService.Object,
        _mailService.Object, _supportMailService.Object,
        _minionCommander.Object,
        _authenticatedDeviceService.Object,
        _cardGatewayConfigurationConsultant.Object,
        _fieldUIService.Object, _validationService.Object);
```

„Tests should be coupled to the behaviour of code and decoupled from the structure of code.”

- Kent Beck

SO…

high cohesion good, high coupling bad?

nothing comes free

the more you disconnect things, the more loosely coupled they become,

the more opportunities for independent failures and bugs

advantage: if you make a change to some logic in one part, the likelihood of breaking the other parts is very small

this helps projects get done faster

- there isn’t a single best design for any system
- there are many poor designs for that same system

- the more experienced programmer, the more design ideas she/he knows
- temptation: put design ideas in the system now, because you'll need them *eventually*

Over-designing:
- feedback from real usage delayed
- adding features more complicated
- adapting the design more difficult

Under-designing:
- increased number of defects
- adding features more complicated
- adapting the design more difficult

need for balance

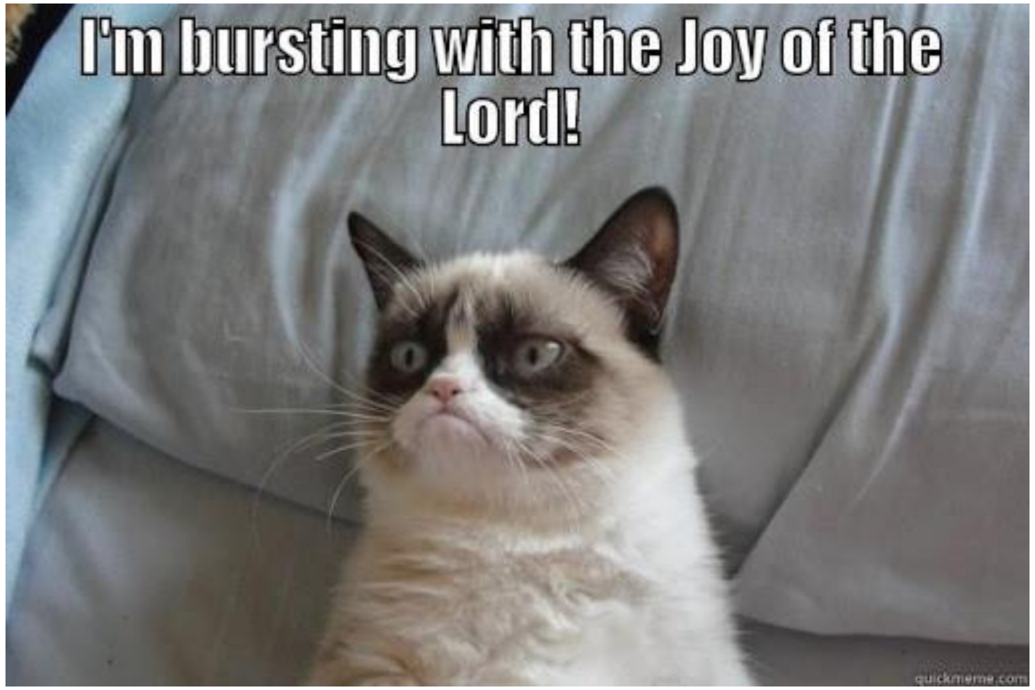
- some design needs to be done in advance of coding
- most design should be done over the life of the project in response to the changing needs of the system and growing understanding of the developers

„**big discovery upfront, small design upfront**”

the:protocol

- elements too large -> each change will be more expensive than it needs to be
- elements too small -> changes will ripple across elements

Design optimization made against the stream of unpredictable changes.



robustness, resilience and scalability?

build more loosely coupled systems -

more event-driven and asynchronous

-> of course, new challenges arise

conclusion

clear responsibility of a module

coherent language inside a module

awareness about dependencies

„Be explicit about what goes in & out” – easy-to-understand interfaces

„Duplication is far cheaper than the wrong abstraction” (**A**void **H**asty **A**bstractions)

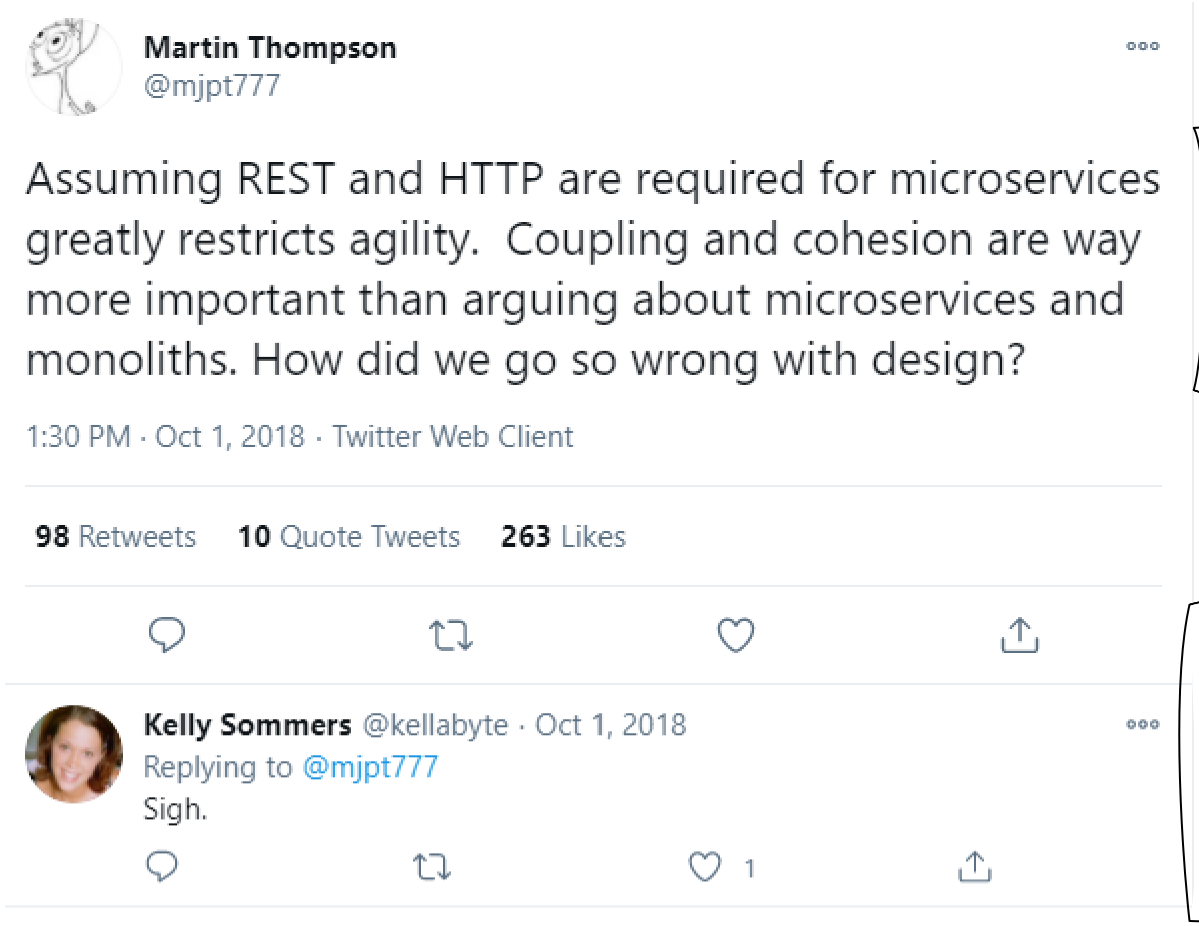
„Make change easy, than make easy change”

„Design is about choice, but that choice should be a deliberate one, and we should be aware of the consequences of our choices.”

Agile Technical Practices Distilled Pedro Moreira Santos, Marco Consolaro and Alessandro Di Gioia

side note: Dan North - Decisions, Decisions

the:protocol



Martin Thompson
@mjpt777

Assuming REST and HTTP are required for microservices greatly restricts agility. Coupling and cohesion are way more important than arguing about microservices and monoliths. How did we go so wrong with design?

1:30 PM · Oct 1, 2018 · Twitter Web Client

98 Retweets **10** Quote Tweets **263** Likes

Kelly Sommers @kellabyte · Oct 1, 2018
Replying to @mjpt777
Sigh.

1

Martin Thompson
@mjpt777

Pasty faced performance gangster - real-logic.co.uk

225 Following **14.6K** Followers

Kelly Sommers
@kellabyte

4x Windows Azure MVP & Former 2x DataStax MVP for Apache Cassandra, Backend brat, big data, distributed diva. Relentless learner. I void warranties.

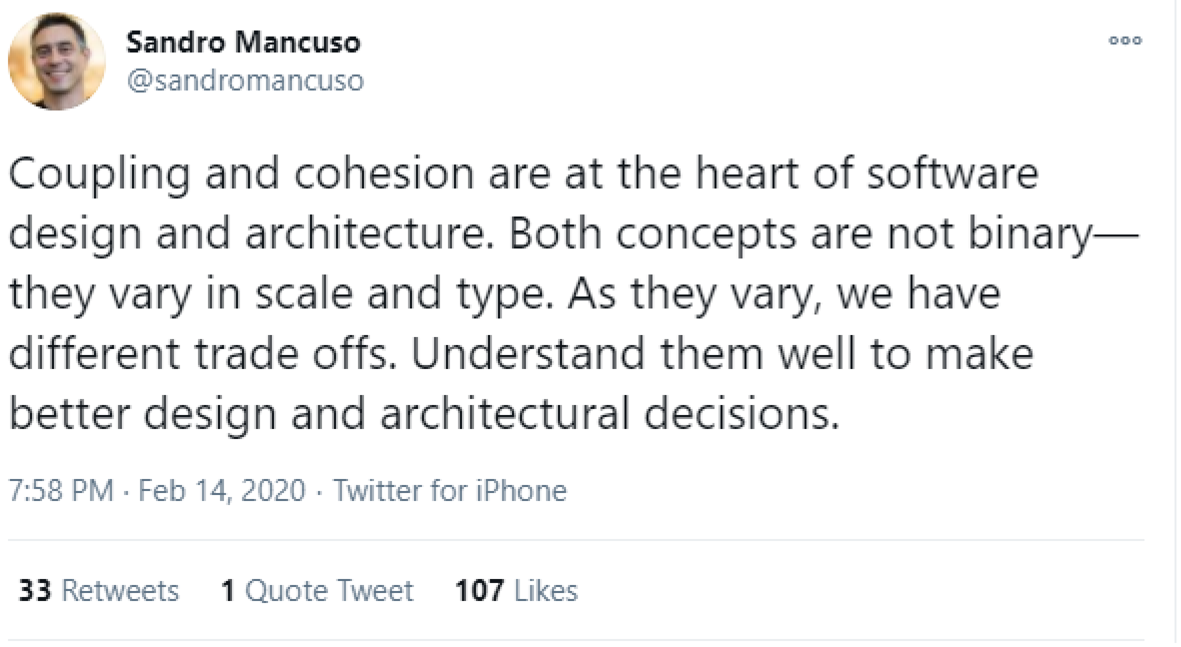
320 Following **43.1K** Followers

the:protocol

Sandro Mancuso
@sandromancuso

Software craftsman | co-founder @codurance | founder of the London Software Craftsmanship Community | author of The Software Craftsman: goo.gl/b9EymU

219 Following 9,770 Followers

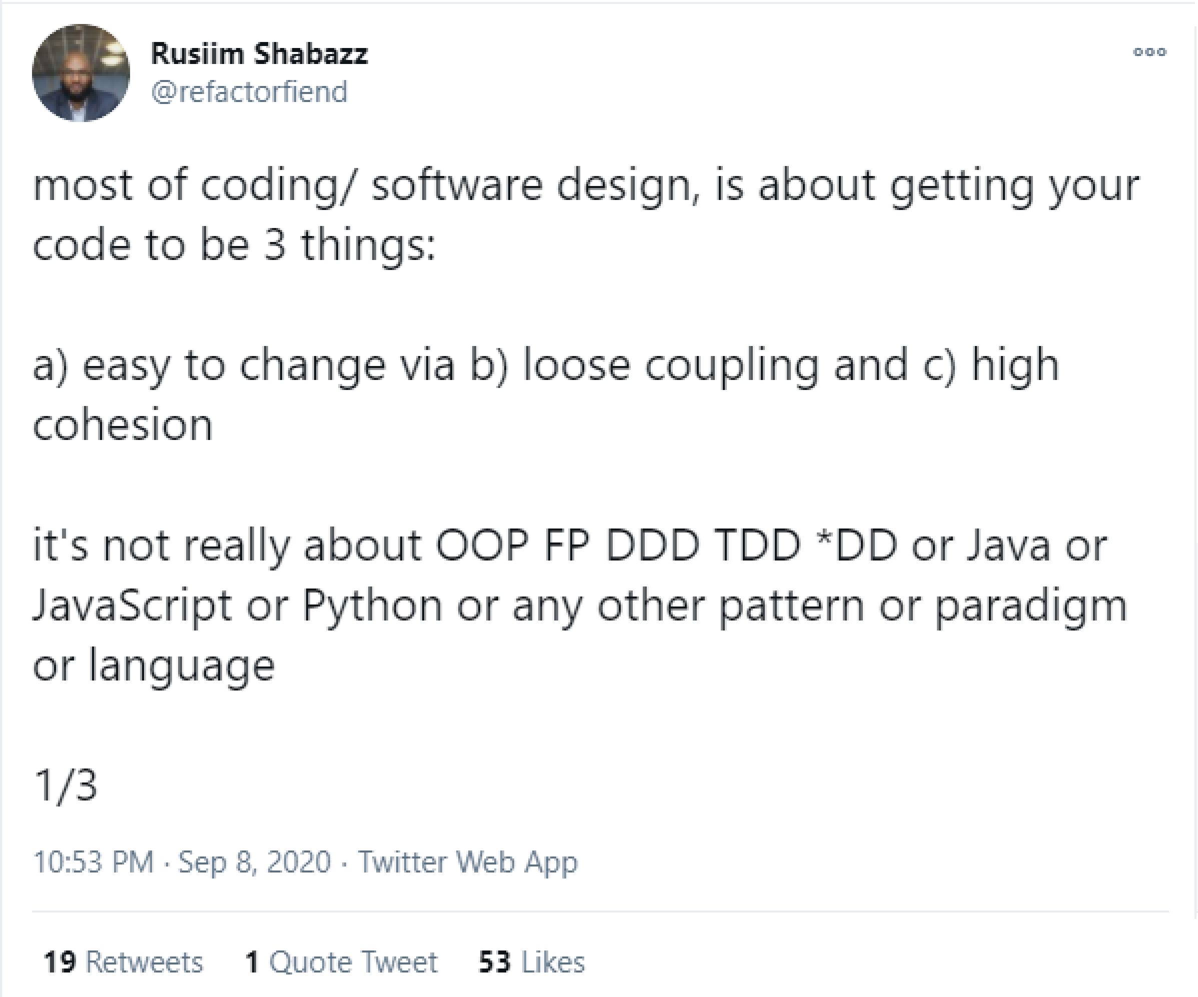


Sandro Mancuso
@sandromancuso

Coupling and cohesion are at the heart of software design and architecture. Both concepts are not binary— they vary in scale and type. As they vary, we have different trade offs. Understand them well to make better design and architectural decisions.

7:58 PM · Feb 14, 2020 · Twitter for iPhone

33 Retweets 1 Quote Tweet 107 Likes



Rusiim Shabazz
@refactorfiend

most of coding/ software design, is about getting your code to be 3 things:

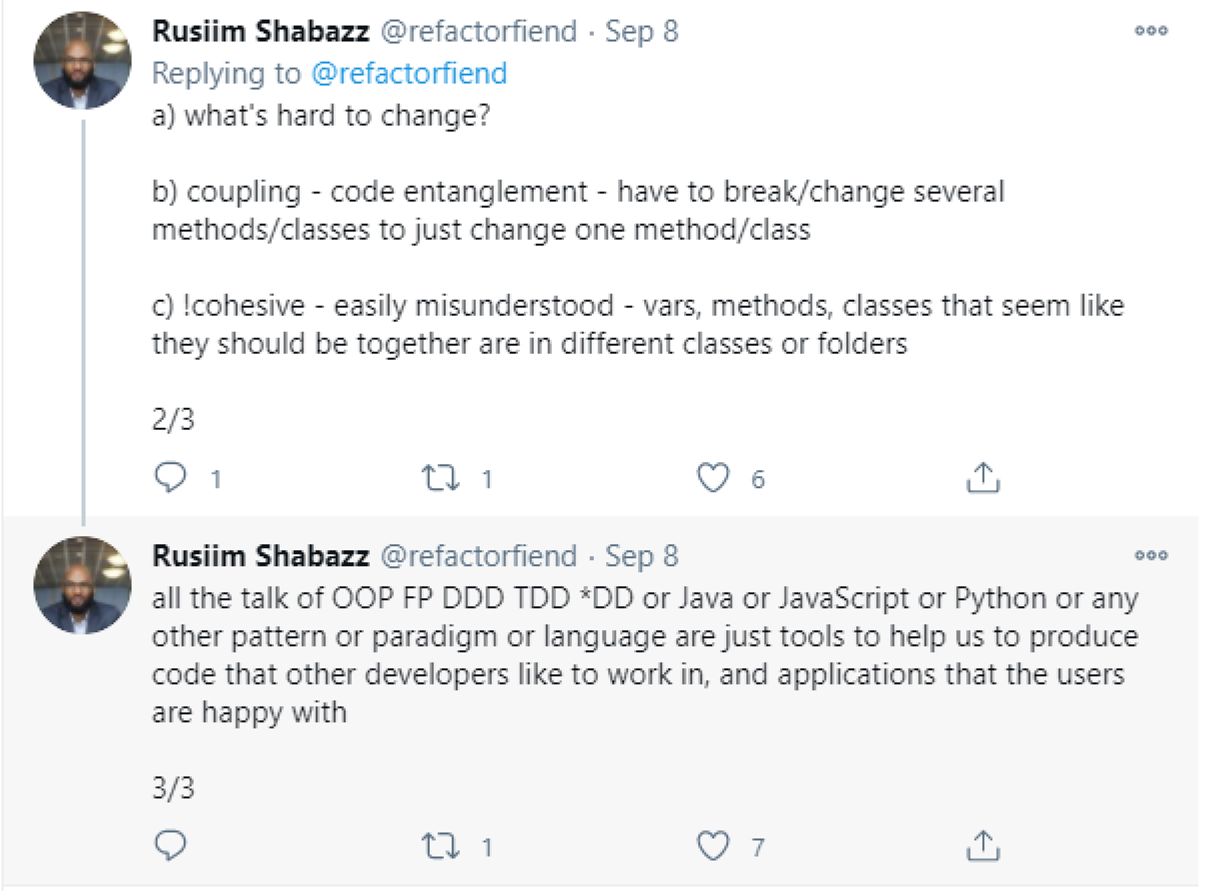
a) easy to change via b) loose coupling and c) high cohesion

it's not really about OOP FP DDD TDD *DD or Java or JavaScript or Python or any other pattern or paradigm or language

1/3

10:53 PM · Sep 8, 2020 · Twitter Web App

19 Retweets 1 Quote Tweet 53 Likes



Rusiim Shabazz @refactorfiend · Sep 8
Replying to @refactorfiend
a) what's hard to change?

b) coupling - code entanglement - have to break/change several methods/classes to just change one method/class

c) !cohesive - easily misunderstood - vars, methods, classes that seem like they should be together are in different classes or folders

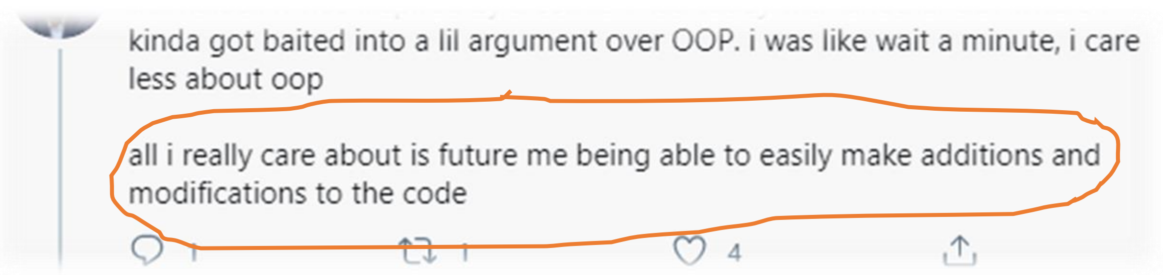
2/3

1 1 6

Rusiim Shabazz @refactorfiend · Sep 8
all the talk of OOP FP DDD TDD *DD or Java or JavaScript or Python or any other pattern or paradigm or language are just tools to help us to produce code that other developers like to work in, and applications that the users are happy with

3/3

1 7



kinda got baited into a lil argument over OOP. i was like wait a minute, i care less about oop

all i really care about is future me being able to easily make additions and modifications to the code

1 1 4

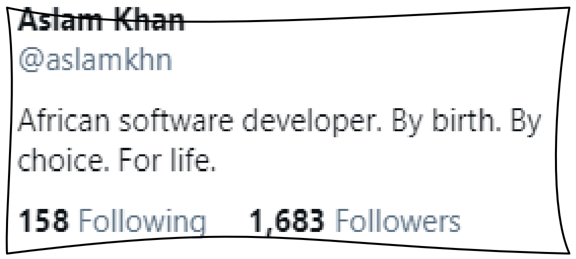
Rusiim Shabazz
@refactorfiend

software lyricist, corporate ghostwriter. java, js, sae, ebonics, patwa spitta 🇺🇸🇯🇲🇻🇨

809 Following 1,067 Followers

Followed by EventModeling, Mob Mentality Show, and 20 others you follow

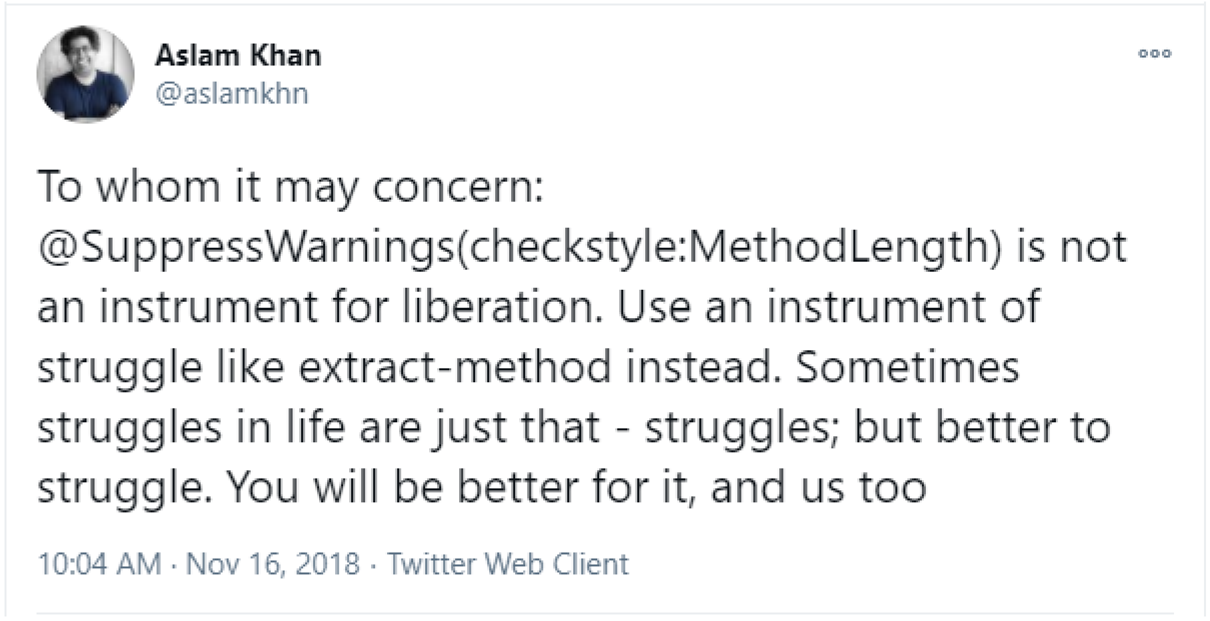
the:protocol



Aslam Khan
@aslamkhn

African software developer. By birth. By choice. For life.

158 Following **1,683** Followers

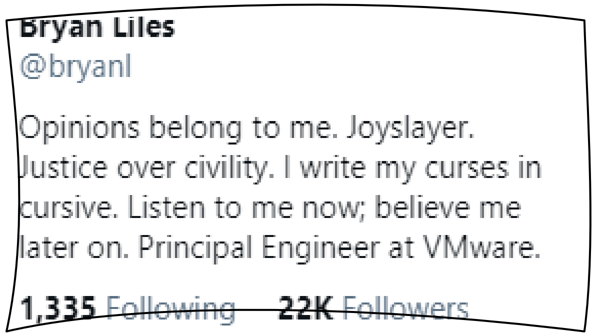


Aslam Khan
@aslamkhn

To whom it may concern: @SuppressWarnings(checkstyle:MethodLength) is not an instrument for liberation. Use an instrument of struggle like extract-method instead. Sometimes struggles in life are just that - struggles; but better to struggle. You will be better for it, and us too

10:04 AM · Nov 16, 2018 · Twitter Web Client

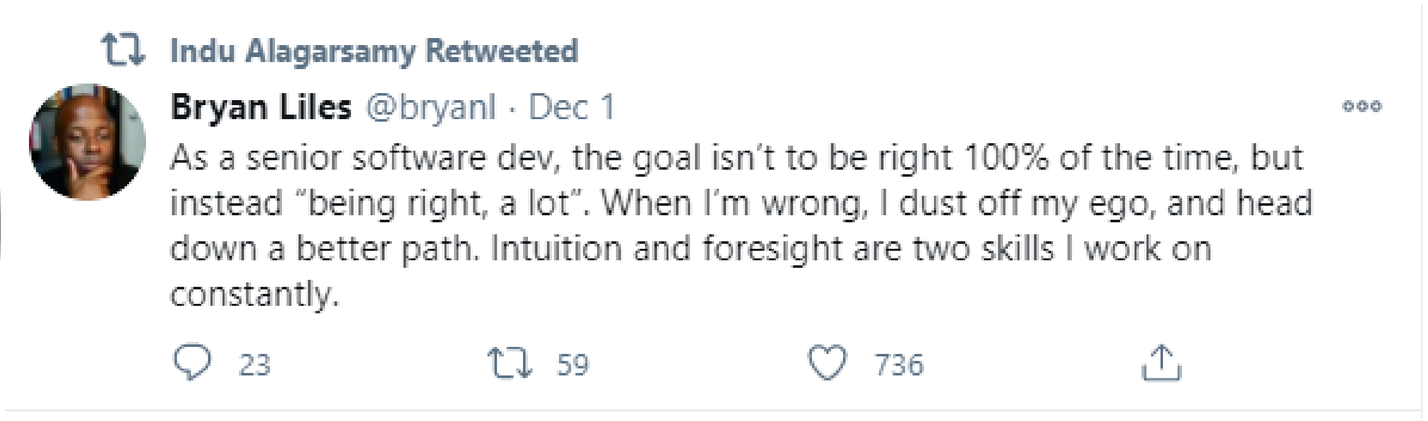
the:protocol



Bryan Liles
@bryanl

Opinions belong to me. Joyslayer. Justice over civility. I write my curses in cursive. Listen to me now; believe me later on. Principal Engineer at VMware.

1,335 Following 22K Followers



Indu Alagarsamy Retweeted

Bryan Liles @bryanl · Dec 1

As a senior software dev, the goal isn't to be right 100% of the time, but instead "being right, a lot". When I'm wrong, I dust off my ego, and head down a better path. Intuition and foresight are two skills I work on constantly.

23 59 736

more on the subject

„**Agile Technical Practices Distilled”** Pedro Moreira Santos, Marco Consolaro and Alessandro Di Gioia

Kent Beck - Cohesion and Coupling

Josh Robb - Connascence & Coupling

https://www.infoq.com/news/2007/12/7-levels-loose-coupling/

https://www.infoq.com/news/2014/04/dahan-eda-loosely-coupled/

https://refactoring.guru/smells/

https://www.destroyallsoftware.com/blog/2014/test-isolation-is-about-avoiding-mocks

the:protocol

questions?

„We're flooding people with information. We need to feed it through a processor. A human must turn information into intelligence or knowledge. We've tended to forget that no computer will ever ask a new question."

- Grace Hopper, Ph.D.

US Navy Rear Admiral

the:protocol

kinga.gazdzinska@pracuj.pl

https://www.linkedin.com/in/gazdzinskak

https://github.com/gazdzinskak

Dziękuję!

the:protocol