Types of Cohesion

https://mrpicky.dev/the-forgotten-realm-of-cohesion/

Coincidental Cohesion

- The first criterion for assigning elements to a specific class could be an actual lack of any criteria.
- It means that elements are grouped totally randomly – a real free-for-all.
- Such classes are created coincidentally and not in the process of design.

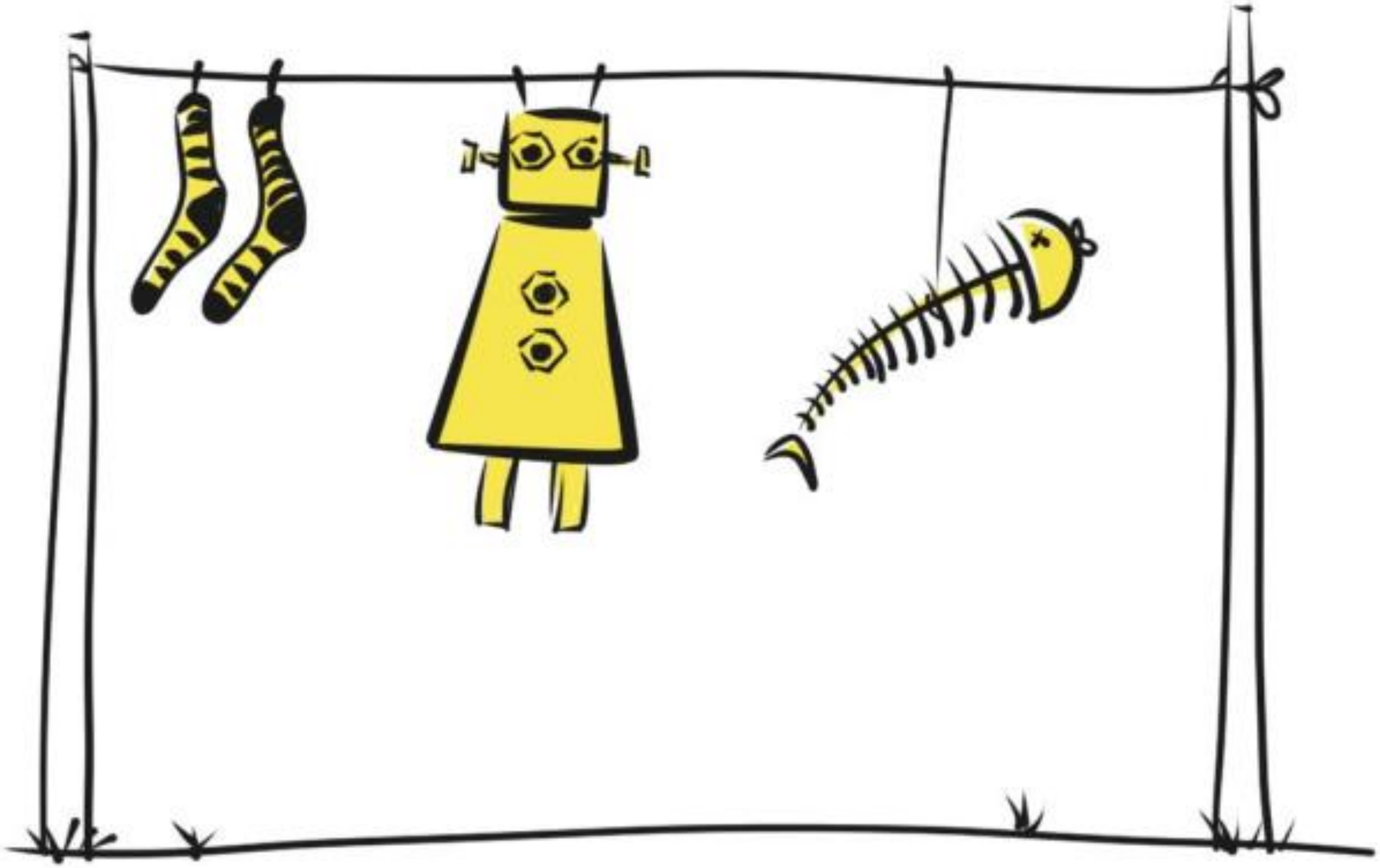
Coincidental Cohesion

- If we were to give this type any number on the scale of cohesion, we would probably give it a zero as there is no cohesion here at all.

Coincidental Cohesion

- Classes with this type of cohesion are quite rare but if we find any, they are probably artifacts of bad modularization.
- Usually, they are created because we find some sequence of methods’ calls in a few different places in our application, and we decide to extract them to a single class.

Coincidental Cohesion – Total randomness in grouping elements into a class



Logical Cohesion

- The elements of a class are grouped because they solve problems from the same category.
- Logically there is something that they have in common even though they have different functionalities.
- A good example here could be mathematical operations, as each of them can do various things, but often they are grouped in some Math or Utils class with… mathematical operations.

Class' elements solve the same type of problems. For example, a class that aggregates all mathematical operations.

$$\underline{y = \left(\frac{2}{x}\right)^2 + \sqrt[3]{x}}$$

Temporal Cohesion

- If logical cohesion is combined with another grouping criterion that is the moment of execution, we will probably get temporal cohesion.

- The best examples of this type of cohesion would be classes/modules that initialize, stop, or clean something.

- Initialization methods are logically connected as they “initialize something”.

- What is more, those methods usually have to be executed at some specific point in time or period.

Temporal cohesion – elements of the class have to be executed within the same period.

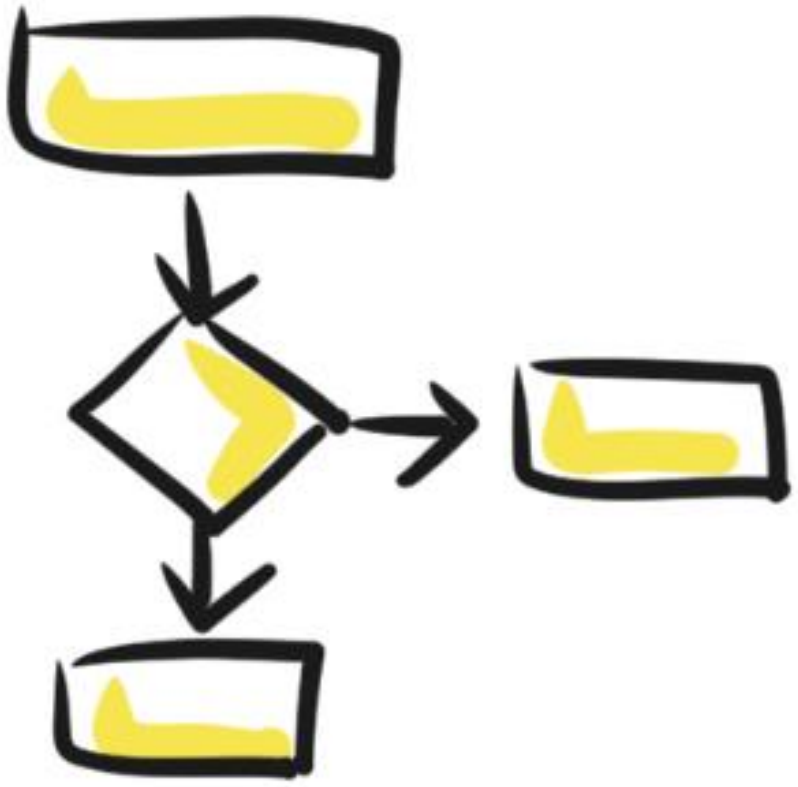
Procedural Cohesion

- The next type is procedural cohesion, and it usually occurs when instead of modelling a domain we focus on an algorithm of execution – the order of steps that have to be executed to get from state “A” to state “B”.
- It may lead us to the approach where we end up modelling the algorithm itself and not the problem that we were supposed to solve.

Procedural Cohesion

- Cohesion in this type is higher as elements are not only grouped because of their moment of execution like in temporal cohesion, but they also have to be executed in a specific order.
- Loops, multiple conditions, and steps in the code can show evidence of procedural cohesion.
- Those are usually classes that, while being analyzed, make us want to take a piece of paper, a pencil and start to draw a block diagram.

Usually used when we model algorithm itself and not the problem that we were supposed to solve



Communicational Cohesion

- The first type of cohesion that focuses on the modelled problem is communicational cohesion.
- The criterion for grouping elements into a class here is that they are communicationally connected, which means that they work on the same input or return the same type of output data.
- Thanks to that, it can be easier to reuse the whole class in different contexts.

Communicational Cohesion

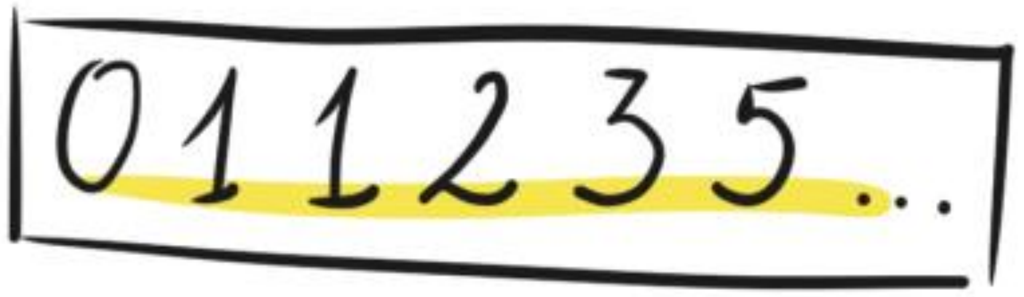
- Classes like this have origins in thinking about operations that can be done on some particular set of data, or on the other hand – operations that have to be executed to get that set of data.

Elements in the class are grouped because they are communicationally connected, so they use the same input data or return the same output data.

Sequential Cohesion

- At the stage when we group elements because of their sequence of data processing, so that the output of one element is at the same time input of the next one we probably have a brush with sequential cohesion.
- There could be only one small issue here.
- If we have such a sequence, we can cut it in different ways.
- It means that created classes can do more than one functionality, or quite the opposite – only some part of the functionality.
- That’s the only reason why in the hierarchy of cohesion types it stands below the last one, which is functional cohesion.

We group elements because of the sequence of data processed by them



Functional Cohesion

- We achieve it when all elements within a class are there because they work together in the best possible way to accomplish the goal – functionality.
- Each processing element of such a class is its integral part and is critical for the functionality of the class.
- The class itself performs no less and no more than one functionality.

Elements within a class are there because they work together in the best possible way to accomplish the goal – functionality.