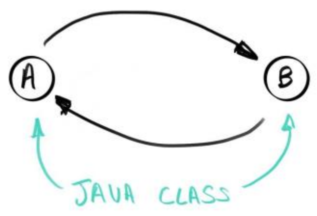
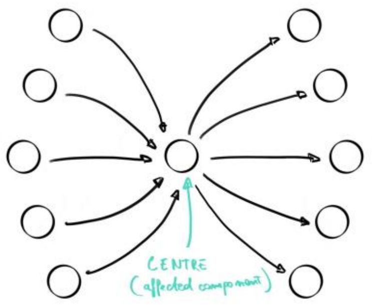
What are architectural smells?

Arcan detects *architectural smells*, software design decisions which negatively impact the maintainaibility, capability of evolve and security of the project. They are symptom of Technical Debt.

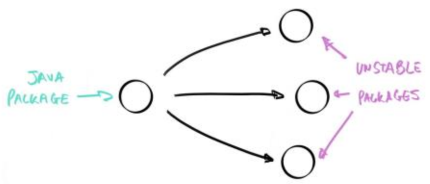
Cyclic Dependency (CD): when two or more architectural components are involved in a chain of relationships. The parts of code affected by CD are hard to release, to maintain and to reuse in isolation. (detected on units and containers)



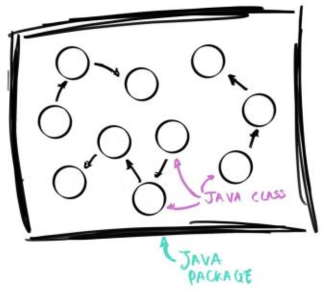
Hub-Like Dependency (HL): when an architectural component has (outgoing and ingoing) dependencies with a large number of other components. The component affected by HL centralizes logic, is a unique point of failure and favors change ripple effects. (detected on units and containers)

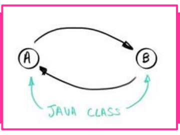


Unstable Dependency (UD): describes an architectural component that depends on other components that are less stable than itself. This can cause a ripple effect of changes in the system. Instability (proneness to change) is measured using R.C. Martin's formula. (detected on containers)



God Component (GC): this smell occurs when an architectural component is excessively large in terms of LOC (Lines Of Code). God components centralize logic and are a sign of low cohesion within the affected component, increasing complexity. (detected on units and containers)





Cyclic Dependency

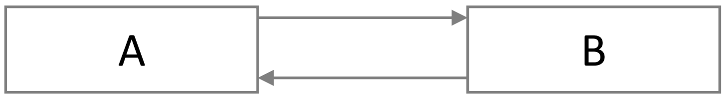
Refactoring – best practices

The main goal when refactoring a cycle is to break one or more dependencies.

Refactoring of Cyclic Dependencies affecting UNITS

To better exemplify the techniques, we will consider a cycle made of two units, A and B. The techniques can be applied to break larger cycles by iterating the refactoring steps.

Figure 1: graphical example of Cyclic Dependency smell

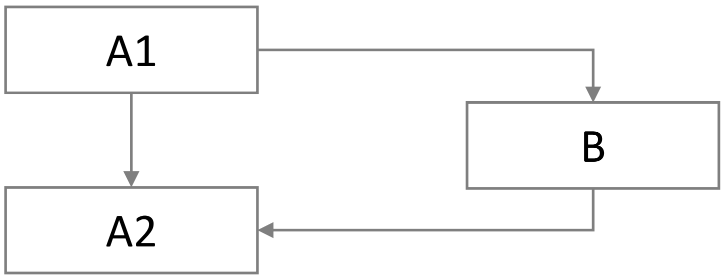


Depending on the characteristics of the dependency to break, different refactoring techniques can be adopted:

- **Move function**: move one or more *functions* (e.g., Java method) into the target unit. This technique is feasible when the function is not invoked within the unit itself and the only problem is that the function is misplaced.

- **Extract unit**: create a new unit and place all the code responsible for the cyclic dependency in it.

Figure 2: example of "Extract unit" refactoring technique



Cyclic Dependency

Refactoring – best practices

- **Create interface:** Introduce an interface for one of the abstractions involved in the cycle. The new interface contains all methods that A calls on B. A only knows the interface that is implemented by B. Mind that this technique can be used if A only *uses* B and does not generate instances of B.

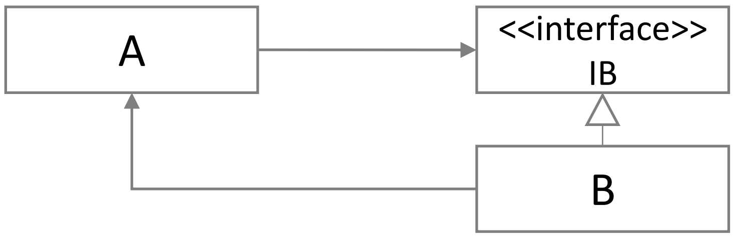


Figure 3: example of "Create interface" refactoring technique

- **Merge units:** if the units involved in the cycle represent a semantically single object, merge the units into a single unit.

Refactoring of Cyclic Dependencies affecting CONTAINERS

In the case of Cyclic Dependencies affecting *containers* (e.g., Java packages), you can face two possibilities:

1. **The cycle is caused by a cycle among units**. In such a case, you can apply the same techniques for breaking cycles among units.
2. **The cycle only exists among packages**. In such a case, it is likely that there is a misplaced unit of function. You must move one of the pieces of code generating the dependency.
    a) Apply move unit from one container to another in the case an entire unit or units are misplaced.
    b) Apply move function from one unit to another in the case one or more functions are misplaced.

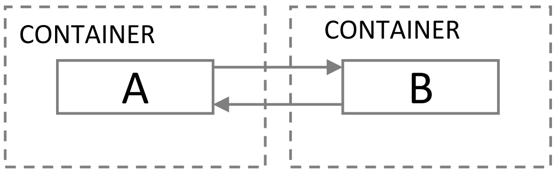


Figure 4: example of container cycle caused by a cycle among units

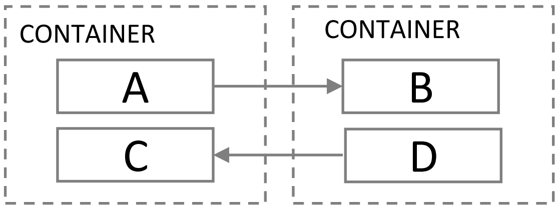
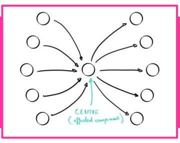


Figure 5: example of container cycle caused by misplaced code

ARCAN



Hub – Like Dependency

Refactoring – best practices

Removing a hub means reassigning the responsibilities and roles of some classes in order to redesign their dependencies. It would ideal to split up the responsibilities of the hub across both new and already existing abstractions, so that the number of dependencies is reduced.

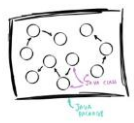
To remove Hub-Like Dependency, the following refactoring techniques can come in handy:

- **Extract method**: Create a new method with a similar body in another class. Either turn the old method into a simple delegation or remove it altogether.
- **Extract abstract unit**: Create an abstract unit and move the common features to the abstract unit.
- **Extract unit**: Create a new unit and move the relevant fields and functions from the old unit into the new unit.

When Arcan detects that a unit is affected by a Hub-Like Dependency, to remove it, you can apply the following steps:

1. Identify the *centre* of the hub, that is the component having a large number of incoming and outgoing dependencies. You can find this information on Arcan.
2. Work on the *centre* using the refactoring techniques mentioned above.
3. Consider the following:
    a) If you are working on a unit, try identify at least two different roles/responsibilities, then, for each one of them, create a new unit by applying "Extract unit". Additionally, you can also extract common functions/fields into a new superclass using "Extract Superclass".
    b) If you are working on a funciton, watch the incoming and the outgoing dependencies used and try to create a cohesive set of responsibilities for the function. Split the function into several different functions, each one with a different responsibility.

To remove the hub, you may also need to move the function into a different unit (you may consider creating a new unit, don not force cohesion) and thus the coupling should decrease.



God Component

Refactoring – best practices

The refactoring of God Components affecting **containers** can be carried out by splitting the content of the container into additional containers. First, you must inspect their dependencies to identify sets of cohesive units to separate. Privilege units that:

- share dependencies with a high "weight" value. This means they are strongly coupled. You can find the *weight* metric on Arcan.
- are involved in Cyclic Dependencies.

Once you identify sets of cohesive units, move them into newly created containers. Mind that you could unintentionally add Cyclic Dependencies among containers while moving.

The presence of a God component affecting containers likely implies the presence of a God Component affecting **units** too.

In such a case, the main aim when refactoring a God component affecting units is to split it up. To do so, you can apply different techniques:

- **Extract function:** often, units affected by God Components contain duplicated code. You must identify the duplicated code and extract a function in this case. Then, use the function in each part of the code where the code was duplicated.
- **Extract unit:** if the affected unit implements different behaviors simultaneously, parts of the unit can be extracted and put into a new, separated unit.

Refactoring best practices for Unstable Dependency smell are coming soon.