Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

Difference Between Cohesion and Coupling

Ask Question

What is the difference between cohesion and coupling?

How can coupling and cohesion lead to either good or poor software design?

What are some examples that outline the difference between the two, and their impact on overall code quality?

oop ooad

edited Jul 10 at 3:07
iliketocode
5,399 4 30 45

asked Jun 21 '10 at 14:01
JavaUser
8,566 35 83 108

1 check it out: msdn.microsoft.com/en-us/magazine/cc947917.aspx – Inv3r53 Jun 21 '10 at 14:03

1 I would like to point out to this article: S.O.L.I.D. Software Development, One Step at a Time. Grz, Kris. – Kris van der Mast Jun 21 '10 at 14:40

1 See also: stackoverflow.com/questions/39946/coupling-and-cohesion
– davidpricedev Sep 8 '15 at 14:04

13 Answers

Cohesion refers to what the class (or module) can do. Low cohesion would mean that the class does a great variety of actions - it is broad, unfocused on what it should do. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.

Example of Low Cohesion:

```
-------------------
| Staff           |
-------------------
| checkEmail()    |
| sendEmail()     |
| emailValidate() |
| PrintLetter()   |
-------------------
```

Example of High Cohesion:

```
----------------------------
| Staff                    |
----------------------------
| -salary                  |
| -emailAddr               |
----------------------------
| setSalary(newSalary)     |
| getSalary()              |
| setEmailAddr(newEmail)   |
| getEmailAddr()           |
----------------------------
```

As for **coupling**, it refers to how related or dependent two classes/modules are toward each other. For low coupled and maintain your code; since classes are closely knit together, making a change could require an entire system revamp.

Good software design has **high cohesion** and **low coupling**.

edited Oct 18 '17 at 21:31
Connor Low
15 3

answered Jun 21 '10 at 14:17
mauris
32.9k 13 82 123

7 I don't see how removing a few methods, and adding a few others increases cohesion. Can someone help here please? – Saket Jain Jan 27 '16 at 9:44

3 the example of low cohesion at the top looks pretty good, I think you accidentally meant to say "high cohession" – relipse Mar 4 '16 at 14:45

17 @SaketJain The Staff class it's not the place where we check, send or validate emails. Those functions should go inside an hypothetical Email class, that's the reason it's a low cohesion. In the second example the Staff class contains only proper information for setting and getting Staff related data. They don't perform actions that should be managed by another class. – Antonio Pantano Oct 14 '16 at 22:09

1 @AntonioPantano Thanks! That does clear it up. :) – Saket Jain Oct 17 '16 at 11:27

1 very nice explanation, thank you – Moustafa Elkady Apr 8 '17 at 11:42

High cohesion *within* modules and low coupling *between* modules are often regarded as related to high quality in OO programming languages.

For example, the code inside each Java class must have high internal cohesion, but be as loosely coupled as possible to the code in other Java classes.

Chapter 3 of Meyer's Object-Oriented Software Construction (2nd edition) is a great description of these issues.

edited Feb 17 '15 at 16:59
nbro
5,517 8 46 92

answered Jun 21 '10 at 14:04
CesarGon
12k 5 46 78

Cohesion is an indication of how related and focused the responsibilities of an software element are.

Coupling refers to how strongly a software element is connected to other elements.

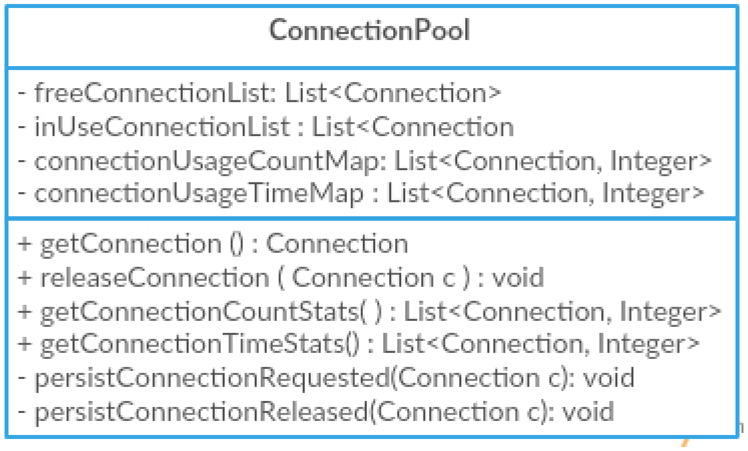
The software element could be class, package, component, subsystem or a system. And while designing the systems it is recommended to have software elements that have **High cohesion** and support **Low coupling**.

Low cohesion results in monolithic classes that are difficult to maintain, understand and reduces re-usablity. Similarly **High Coupling** results in classes that are tightly coupled and changes tend not be non-local, difficult to change and reduces the reuse

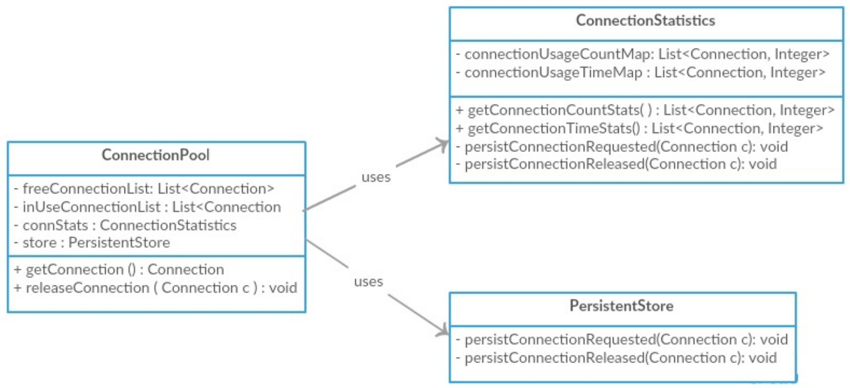
We can take a hypothetical scenario where we are designing an typical monitor-able `ConnectionPool` with the following requirements. Note that, it might look too much for a simple class like `ConnectionPool` but the basic intent is just to demonstrate **low coupling** and **high cohesion** with some simple example and I think should help.

1. support getting a connection
2. release a connection
3. get stats about connection vs usage count
4. get stats about connection vs time
5. Store the connection retrieval and release information to a database for reporting later.

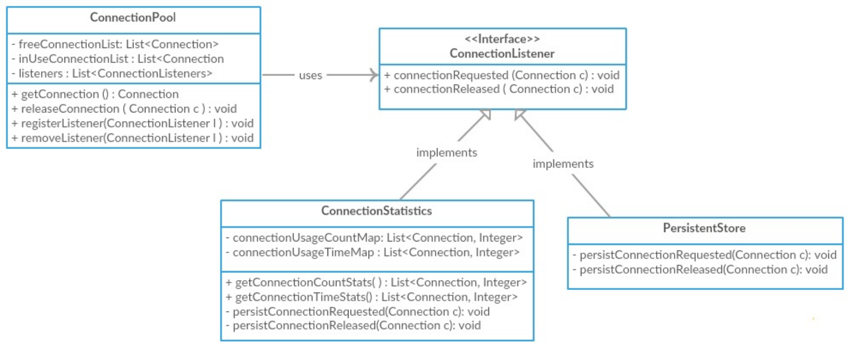
With **low cohesion** we could design a `ConnectionPool` class by forcefully stuffing all this functionality/responsibilities into a single class as below. We can see that this single class is responsible for connection management, interacting with database as well maintaining connection stats.


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

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>
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void
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

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>
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void
PersistentStore
- persistConnectionRequested(Connection c): void
- persistConnectionReleased(Connection c): void

Note/Word or Caution: For this simple scenario it may look like an overkill but if we imagine a real-time scenario where our application needs to interact with multiple third party services to service could result in changes to our code at multiple places, instead we could have `Facade` that interacts with these multiple services internally and any changes to the services become local to the `Facade` and enforce low coupling with the third party services.

answered Jan 6 '16 at 13:40

Madhusudana Reddy Sunnapu
5,600 2 8 19

3 Excellent answer! If possible, could you use some other example? Connection Pooling might not be clear to everyone. Regardless, it really helped me. So thanks! – Saket Jain Jan 27 '16 at 9:56

how does using the ConnectionListener Interface help in reducing coupling?Can you provide an example that is easier to understand. – abhishek gupta Apr 30 '16 at 19:12

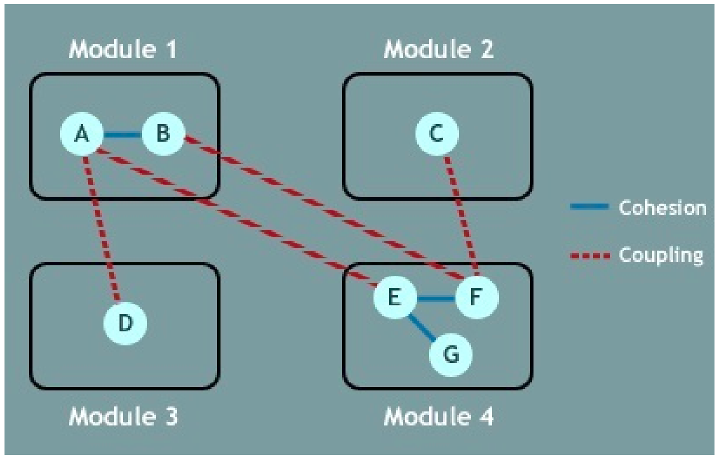
1 @abhishekgupta In this example you might have noticed that we have used observer pattern to achieve low/loose coupling. Going through this would help How does Observer create loosely-coupled design? – Madhusudana Reddy Sunnapu May 1 '16 at 4:36

Thank you @MadhusudanaReddySunnapu It helped me to understand very well. – Siva R Sep 11 '17 at 6:30

Cohesion is the indication of the relationship **within** a module.

Coupling is the indication of the relationships **between** modules.

1. **Cohesion**
2. **Coupling**



Cohesion

- Cohesion is the indication of the relationship within module.
- Cohesion shows the module's relative functional strength.
- Cohesion is a degree (quality) to which a component / module focuses on the single thing.
- While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.
- Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility. Cohesion is Intra – Module Concept.

Coupling

- Coupling is the indication of the relationships between modules.
- Coupling shows the relative independence among the modules.
- Coupling is a degree to which a component / module is connected to the other modules.
- While designing you should strive for low coupling i.e. dependency between modules should be less
- Making private fields, private methods and non public classes provides loose coupling.
- Coupling is Inter -Module Concept.

check this link

edited Sep 17 at 10:46

answered Feb 12 '17 at 14:41

Buddhika Alwis
612 8 18

Increased cohesion and decreased coupling do lead to good software design.

Cohesion partitions your functionality so that it is concise and closest to the data relevant to it, whilst decoupling ensures that the functional implementation is isolated from the rest of the system.

Decoupling allows you to change the implementation without affecting other parts of your software.

Cohesion ensures that the implementation more specific to

The most effective method of decreasing coupling and increasing cohesion is **design by interface**.

That is major functional objects should only 'know' each other through the interface(s) that they implement. The implementation of an interface introduces cohesion as a natural consequence.

Whilst not realistic in some senarios it should be a design goal to work by.

Example (very sketchy):

```
public interface IStackoverFlowQuestion
      void SetAnswered(IUserProfile user);
      void VoteUp(IUserProfile user);
      void VoteDown(IUserProfile user);
}

public class NormalQuestion implements IStackoverflowQuestion {
      protected Integer vote_ = new Integer(0);
      protected IUserProfile user_ = null;
      protected IUserProfile answered_ = null;

      public void VoteUp(IUserProfile user) {
           vote_++;
           // code to ... add to user profile
      }

      public void VoteDown(IUserProfile user) {
          decrement and update profile
      }

      public SetAnswered(IUserProfile answer) {
           answered_ = answer
           // update u
      }
}

public class CommunityWikiQuestion implements IStackoverflowQues
     public void VoteUp(IUserProfile user) { // do not update pr
     public void VoteDown(IUserProfile user) { // do not update
     public void SetAnswered(IUserProfile user) { // do not upda
```

Some where else in your codebase you could have a module that processes questions regardless of what they are:

```
public class OtherModuleProcessor {
    public void Process(List<IStackoverflowQuestion> questions)
       ... process each question.
    }
}
```

edited Jun 21 '10 at 14:49

answered Jun 21 '10 at 14:25

Adrian Regan
2,110 11 11

best explanation of **Cohesion** comes from Uncle Bob's Clean Code:

*Classes should have a small number of instance variables. Each of the methods of a class should manipulate one or more of those variables. **In general the more variables a method manipulates the more cohesive that method is to its class**. A class in which each variable is used by each method is maximally cohesive.*

*In general it is neither advisable nor possible to create such maximally cohesive classes; on the other hand, **we would like cohesion to be high**. When cohesion is high, it means that the methods and variables of the class are co-dependent and hang together as a logical whole.*

The strategy of keeping functions small and keeping parameter lists short can sometimes lead to a proliferation of instance *separate the variables and methods into two or more classes such that the new classes are more cohesive.*

answered May 16 '15 at 13:56

Cătălin Rădoi
1,091 13 25

Cohesion in software engineering is the degree to which the elements of a certain module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is.

Coupling in simple words, is how much one component (again, imagine a class, although not necessarily) knows about the inner workings or inner elements of another one, i.e. how much knowledge it has of the other component.

I wrote a blog post about this, if you want to read up in a little bit more details with examples and drawings. I think it answers most of your questions.

edited Apr 16 '15 at 7:24

answered Apr 8 '15 at 19:11

TheBoyan
5,630 3 34 55

Cohesion (Co-hesion) : **Co** which means **together**, **hesion** which means **to stick**. The System of sticking together of particles of different substances.

For real-life example:

img Courtesy

> Whole is Greater than the Sum of the Parts -Aristotle.

- **Cohesion** is an ordinal type of measurement and is usually described as “high cohesion” or “low cohesion”. Modules with high cohesion tend to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. In contrast, low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand. wiki
- **Coupling** is usually contrasted with **cohesion**. Low coupling often correlates with high cohesion, and vice versa. Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability. wiki

answered Jan 27 at 2:31
Premraj
28.3k 10 147 115

I think the differences can be put as the following:

- Cohesion represents the degree to which a part of a code base forms a logically single, atomic unit.
- Coupling represents the degree to which a single unit is independent from others.
- It’s impossible to archive full decoupling without damaging cohesion, and vice versa.

In this blog post I write about it in more detail.

edited Dec 17 '15 at 18:24
user663031

answered Sep 2 '15 at 11:57
Vladimir
1,043 1 11 27

Cohesion is an indication of the relative functional strength of a module.

- A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing.
- Conventional view:

  the “single-mindedness” of a module
- OO view:

  cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself

□Functional

□Layer

□Communicational

□Sequential

□Procedural

□Temporal

□utility

Coupling is an indication of the relative interdependence among modules.

- Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.
- Conventional View : The degree to which a component is connected to other components and to the external world
- OO view: a qualitative measure of the degree to which classes are connected to one another
- Level of coupling

  □Content

  □Common

  □Control

  □Stamp

  □Data

  □Routine call

External #

edited Jan 13 '16 at 2:35

answered Dec 17 '15 at 18:19

zee
2,831 1 24 29

Coupling = interaction / relationship between two modules...
Cohesion = interaction between two elements within a module.

A software is consisting of many modules. Module consists of elements. Consider a module is a program. A function within a program is a element.

At run time, output of a program is used as input for another program. This is called module to module interaction or process to process communication. This is also called as Coupling.

Within a single program, output of a function is passed to another function. This is called interaction of elements within a module. This is also called as Cohesion.

Example:

Coupling = communication in between 2 different families...
Cohesion = communication in between father-mother-child within a family.

edited Dec 4 '16 at 3:19

answered Mar 5 '16 at 17:20

1 So how do you **explain them in context of their impact on a software ?** – Itban Saeed Nov 19 '16 at 19:23

A software is consisting of many modules. Module consists of elements. Consider a module is a program. A function within a program is a element. – Dipankar Nalui Dec 4 '16 at 3:12

Simply put Cohesion means that a class should represent a single concept.

The public interface of a class is cohesive if all the class features are related to the concept that the class represents. For example - Instead of having CashRegister class having CashRegister and Coin features cohesion makes it into 2 classes - CashRegister and Coin class

In coupling one class depends on another as it uses the objects of the class.

The problem with high coupling is that it can create side effects. One change in one class could cause an unexpected error in the other class and could break the whole code.

Generally High Cohesion and low coupling is considered high quality OOP

answered Sep 23 at 18:06

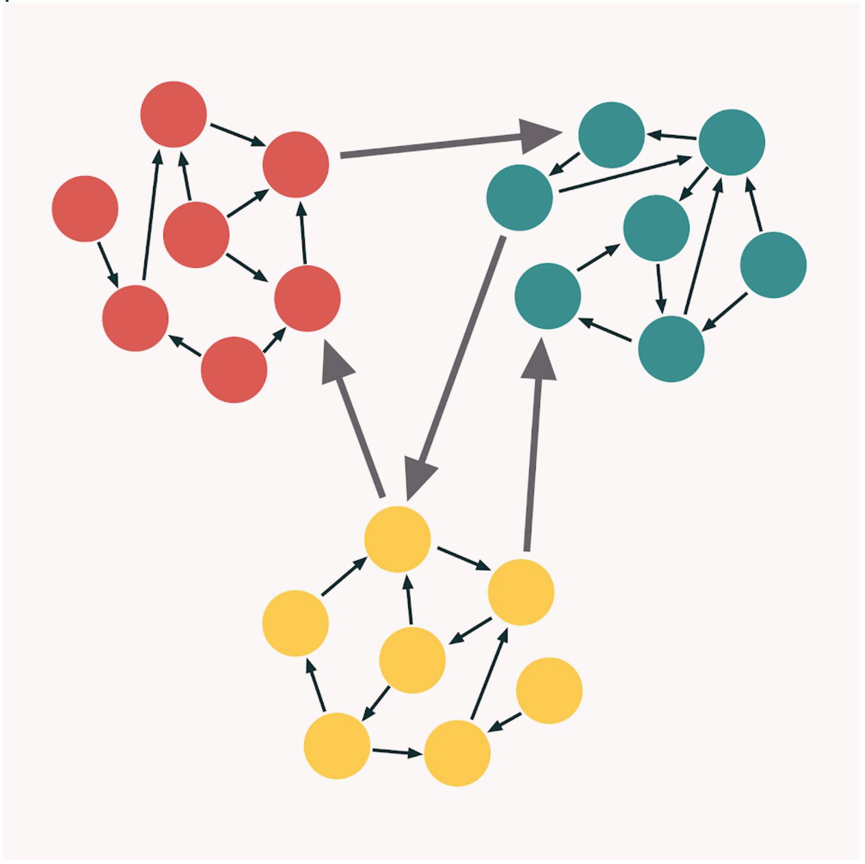
Varun Joshi
24 4

simply, **Cohesion** represents the degree to which a part of a code base forms a logically single, atomic unit. **Coupling**, on the other hand, represents the degree to which a single unit is independent from others. In other words, it is the number of

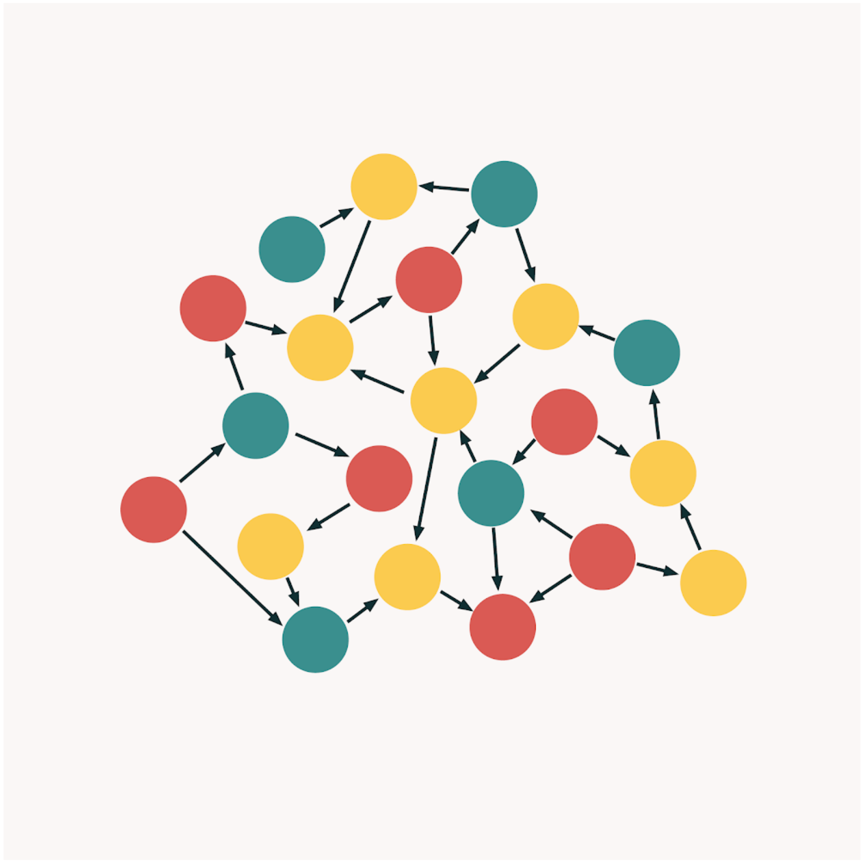
In essence, high cohesion means keeping parts of a code base that are related to each other in a single place. Low coupling, at the same time, is about separating unrelated parts of the code base as much as possible.

Types of code from a cohesion and coupling perspective:

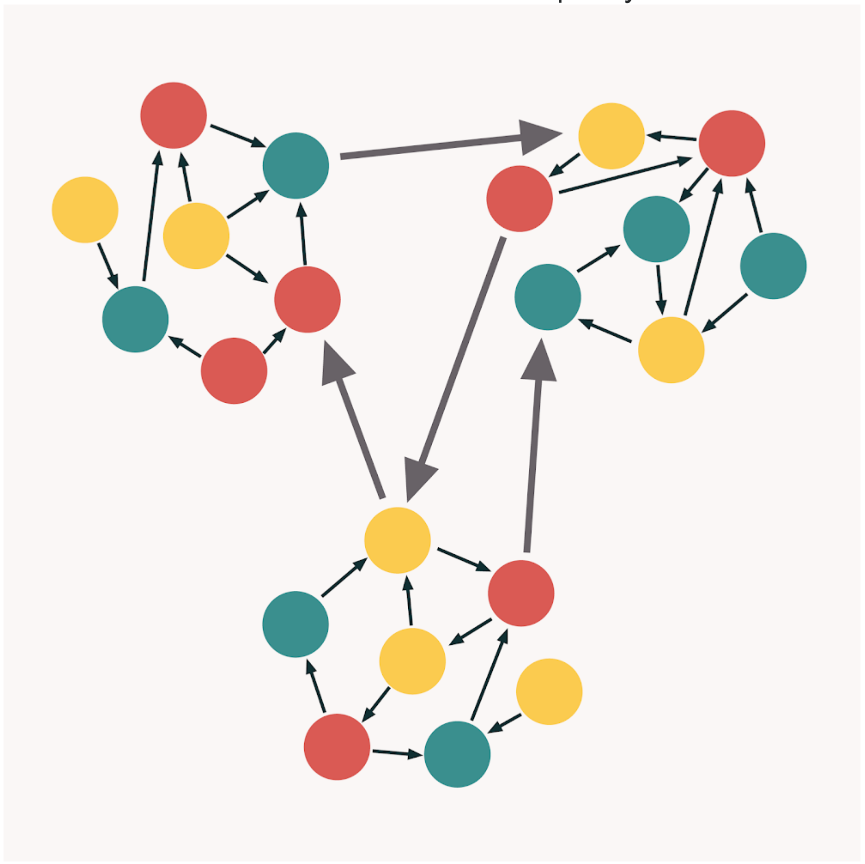
Ideal is the code that follows the guideline. It is loosely coupled and highly cohesive. We can illustrate such code with this picture:



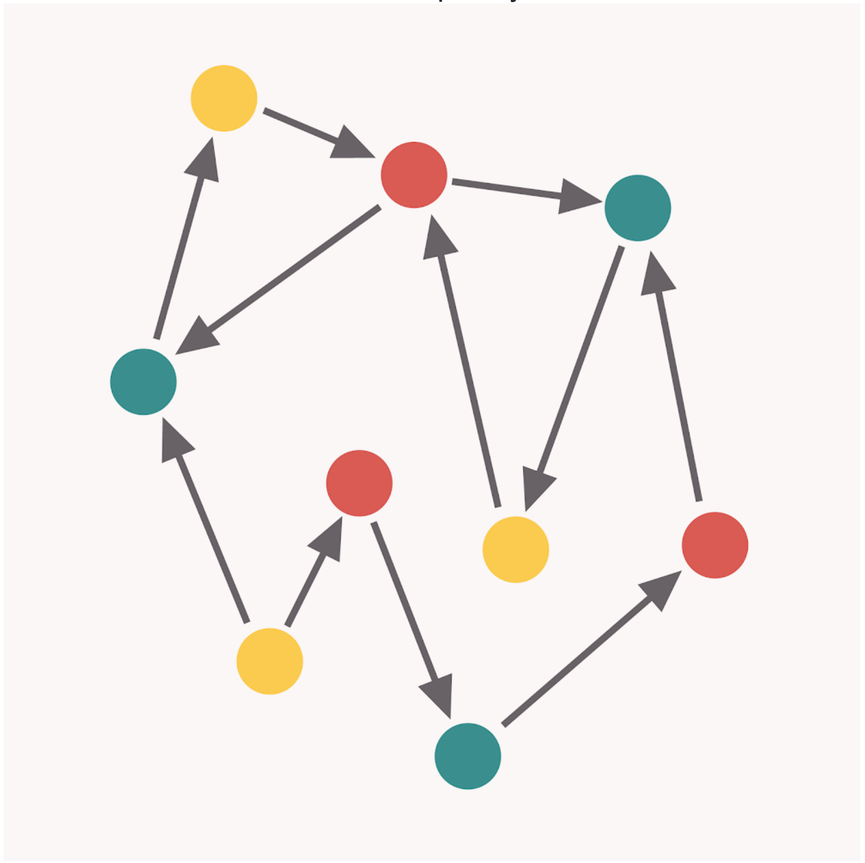
God Object is a result of introducing high cohesion and high coupling. It is an anti-pattern and basically stands for a single piece of code that does all the work at once:



poorly selected takes place when the boundaries between different classes or modules are selected poorly



Destructive decoupling is the most interesting one. It sometimes occurs when a programmer tries to decouple a code base so much that the code completely loses its focus:



read more here

answered Oct 13 at 22:00

Alireza Rahmani Khalili
506 9 16

protected by mauris Jun 29 '16 at 12:50

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, Would you like to answer one of these unanswered questions instead?