Connascence

- What
- Dimensions
- Types
- Exercise

Connascence: What?

Definition: ***The birth and growth of two or more things at the same time***

In software development, we can say that:

Two or more elements (fields, methods, classes, parameters, variables, but also build steps, procedures of any kind, etc.) are connascent if a change in one element would require also a change in the others in order for the system to keep working correctly.

So, connascence generalizes the ideas of Cohesion and Coupling, combining them in a more exhaustive classification using a proper taxonomy

Connascence: Dimensions

Every case of Connascence can be measured over three dimensions: *degree, locality* and *strength*

- Degree: is the size of the impact as estimated by the number of occurrences and the number of entities it affects. The higher the degree, the higher the pain when a modification is needed

- Locality: is the closeness of the affected entities in respect to each other in terms of abstraction. Connascence of entities very far apart from each other is often an indication of design or architectural pitfalls.

- Strength: is the likelihood that Connascence requires compensating changes in connascent elements and how hard it would be to make those changes. The stronger the form of Connascence, the more difficult and costly it would be to refactor the elements in the relationship.

Connascence: Types

Connascence: Examples

- **Connascence of Name** for the fields _hour, _minute, _second (3 appearances each)
- **Connascence of Name** for the parameters hour, minute, second (2 appearances each)
- **Connascence of Type** for the parameters hour, minute and second and respective fields _hour, _minute, _second (2 appearances each)

```
1  public class Time {
2      int _hour;
3      int _minute;
4      int _second;
5
6      public Time(int hour, int minute, int second){
7          _hour = hour;
8          _minute = minute;
9          _second = second;
10     }
11
12     public string Display(){
13         return _hour + “:” + _minute + “:” + _second + “:”;
14     }
15 }
```

Connascence: Examples

```
1  public class NotificationSystem {
2      public void SendEmail(string recipient, string sender, string message) {
3          Email email = new Email();
4          email.To(recipient);
5          email.From(sender);
6          email.Body(message);
7          _smtpService.SendEmail(email);
8      }
9  }
10
11 _notificationSystem.SendEmail("recipient@email.com", "sender@email.com", "text");
```

- **Connascence of Position:** occurs when multiple components must be adjacent or must appear in a particular order – often when passed within a positional structure like an array or a tuple. It is the strongest form of static Connascence and must be carefully avoided, especially for higher degrees.

Connascence: Examples - From Position to Type

```
1  public class NotificationSystem {
2      public void SendEmail(Recipient recipient, Sender sender, Message message) {
3          Email email = new Email();
4          email.To(recipient.Address);
5          email.From(sender.Address);
6          email.Body(message.Body);
7          _smtpService.SendEmail(email);
8      }
9  }
```

```
11 public class Recipient {
12     public Address => _address;
13     private string _address;
14
15     public Recipient(string address){
16         _address = address;
17     }
18 }
19
20 public class Sender {
21     public Address => _address;
22     private string _address;
23
24     public Sender(string address){
25         _address = address;
26     }
27 }
28
```

```
29 public class Message {
30     public Body => _body;
31     private string _body;
32
33     public Message(string body){
34         _body = body;
35     }
36 }
37
```

```
38 _notificationSystem.SendEmail(
39     new Recipient("recipient@email.com"),
40     new Sender("sender@email.com"),
41     new Message("notification text"));
```

Connascence: Examples - Value connascence problem

```
1  public class Time {
2      int _hour;
3      int _minute;
4      int _second;
5
6      public Time(int hour, int minute, int second){
7          _hour = hour;
8          _minute = minute;
9          _second = second;
10     }
11
12     public string Display(){
13         return _hour + “:” + _minute + “:” + _second + “:”;
14     }
15 }
```

```
1  var myTime = new Time (27, 76, 82);
```

What happens with the presented code when setting a new Time?

Connascence: Examples

- **Connascence of Value:** The real issue with CoV is that it is discoverable only at runtime, so we should try to minimize the response time of the negative feedback and validate the input as soon as possible. In this way, we make sure that this class participates in the information flow if – and only if – its state is valid.

```
1  public class Time {
2      int _hour;
3      int _minute;
4      int _second;
5
6      public Time(int hour, int minute, int second){
7          _hour = hour;
8          _minute = minute;
9          _second = second;
10         Validate()
11     }
12
13     public string Display(){
14         return _hour + “:” + _minute + “:” + _second + “:”;
17     private void Validate(){
18         if(_hour < 0 || _hour > 23)
19             throw new InvalidHourException();
20         if(_minute < 0 || _minute > 59)
21             throw new InvalidMinuteException();
22         if(_second < 0 || _second > 59)
23             throw new InvalidSecondException();
24     }
25 }
```

Connascence: Examples

- **Connascence of Meaning:** This happens when two or more components must agree on the meaning of specific values, hence using a convention. In this example, we can see that we present a checkbox with values which need to be interpreted in the controller.

```
<input type='checkbox' name='transport' value='1' />
I travel by bike <br />

<input type='checkbox' name='transport' value='2' />
I travel by car <br />

<input type='checkbox' name='transport' value='3' />
I travel by train <br />

<input type='checkbox' name='transport' value='4' />
I travel by bus <br />
```

```
private SetTransport(string transport){
    switch(transport){
        case "1":
            AddBike();
        case "2":
            AddCar();
        case "3":
            AddTrain();
        case "4":
            AddBus();
        break;
    }
}
```

Connascence: Examples

- **Connascence of Algorithm:** This happens when two or more components must agree on using a particular algorithm.

What will happen if at some point we need to change the algorithm? We'll have to change it in two places. In our case "sum % 10" is our simple algorithm, which is repeated in two places.

```
1  AddChecksum("32143")    =>   "321437"
2  Check("321437")         =>   true
3  Check("321432")         =>   false
```

```
1  public string AddChecksum(string inputData){
2      var sum = SumCharsOf(inputData);
3      var difference = sum % 10;
4      return inputData + difference;
5  }
6
7  public bool Check(string inputData){
8      var sum = SumCharsOf(inputData);
9      return sum % 10 == 0;
10 }
```

A better solution would be to encapsulate the algorithm in its own abstraction layer, then change it only once

```
1  public string AddChecksum(string inputData){
2      return inputData + Checksum(inputData);
3  }
4
5  public bool Check(string inputData){
6      return Checksum(inputData) == 0;
7  }
8
9  private int Checksum(string inputData){
10     var sum = SumCharsOf(inputData);
11     return sum % 10;
12 }
```

Connascence: Examples

- **Connascence of Execution Order:** It happens when the caller of a component must have some unexpressed knowledge about the correct order of the methods to be called.

What happens in this case if we invert the order and call **Archive** before **SendToCustomer**? If it fails, then we have Connascence of Execution Order

```
1  public void SendReceipts(){
2      var receiptSender = new ReceiptSender();
3      var receiptId = NextUnsentReceiptId();
4
5      while(receiptId != null){
6          receiptSender.SendToCustomer(receiptId);
7          receiptSender.Archive(receiptId);
8
9          receiptId = NextUnsentReceiptID();
10     }
11 }
```

Connascence: Examples - Execution Order

If this is the only usage of ReceiptSender, we have a leaking abstraction in its behavior. The **Archive** method is meant to always be called after the **SendToCustomer**, so really there's no need to have both method public. Instead, one method should be enough and the correct execution order could be hidden inside

```
1  public void SendReceipts(){
2      var receiptSender = new ReceiptSender();
3      var receiptId = NextUnsentReceiptId();
4
5      while(receiptId != null){
6          receiptSender.SendToCustomer(receiptId);
7          receiptSender.Archive(receiptId);
8
9          receiptId = NextUnsentReceiptID();
10     }
11 }
```

Connascence: Examples

- **Connascence of Timing**: It happens when the success of two or more calls depends on the timing of when they occur. A classic example would be **race conditions**. In this example, we see our test relying on an artificial 1000ms sleep in order to wait for the message to be delivered. But this doesn't happen in real life

```
1  [TestFixture]
2  public class ServiceBusMessageHandler_Should{
3      var messageHandler = new MessageHandler();
4
5      [Test]
6      public void ReceiveMessages(){
7          var expectedMessage = new TestMessage();
8          SendMessageToIntegrationTestBus(expectedMessage);
9          Thread.Sleep(1000);
10
11         var receivedMessage = messageHandler.ReadLastMessage();
12
13         Assert.That(receivedMessage, Is.Equal(expectedMessage))
14     }
15 }
```

Connascence: Examples - Timing

In this example we've relied on events (OnNewMessage) to be able to simulate a more real-life example. The awaiter component expects an event within 1000ms or else it will fail.
Typescript provides the same type of behavior, especially when dealing with UI testing where you need to wait for some elements to be drawn.

```
1  [TestFixture]
2  public class ServiceBusMessageHandler_Should{
3      var awaiter = new AutoResetEvent(false);
4      var messageHandler = new MessageHandler();
5      messageHandler.OnNewMessage += ()=>awaiter.Set();
6
7      [Test]
8      public void ReceiveMessages(){
9          var expectedMessage = new TestMessage();
10         SendMessageToIntegrationTestBus(expectedMessage);
11         awaiter.WaitOne(1000);
12
13         var receivedMessage = messageHandler.Read();
14
15         Assert.That(receivedMessage, Is.Equal(message))
16     }
17 }
```

Connascence: Examples

- **Connascence of Identity:** This happens when one or more components must reference exactly one particular instance of another entity to work correctly.

What will happen in this case in a distributed environment? Have you experienced something like this before?

```
1  public class GlobalCounter{
2      int count = 0;
3
4      public void Increment(){
5          count++;
6      }
7
8      public int CurrentCount(){
9          return count;
10     }
11 }
12
13 public class Controller{
14     GlobalCounter _counter;
16     public Controller(GlobalCounter counter){
17         _counter = counter;
18     }
19
20     public ActionResult Home(){
21         _counter.Increment();
22     }
23 }
```

Connascence: Examples

- **Connascence of Manual Task:** This is the worst type of Connascence of all and it involves manual tasks that extend even beyond the code itself.
  - Have you found yourself dealing with a process in which, each time you add something new you need to also configure a lot of other files and variables in order to handle properly this new element?

Connascent Mars Rover

One more time, let’s revisit the Mars Rover, but this time, with Connascent elements. Can you spot them all and improve them?