# Programming task

I have been given a programming task from a Scandinavian telecompany in order to assess my technical level of expertise in programming. This is seen more and more often in the IT-industry. Such a task is by some referred to test 11. In [the Joel Test](http://www.joelonsoftware.com/articles/fog0000000043.html) “Do new candidates write code during their interview?”

The team I’m going to join (Hopefully) are using C#, asp.net MVC, NHibernate, Specflow, xUnit and StructureMap.

Contents

[Programming task 1](#_Toc377569724)

[Domain Layer 2](#_Toc377569725)

[Object Oriented Analysis and Design 2](#_Toc377569726)

[6](#_Toc377569727)

[Behavior Driven Development (BDD) 7](#_Toc377569728)

[Infrastructure 9](#_Toc377569729)

[StructureMap 9](#_Toc377569730)

[Presentation Layer and making the whole website 9](#_Toc377569731)

[Conclusion 10](#_Toc377569732)

## Problem definition

The description of the problem domain goes like this:

Assume you work in the development department in a leading telecompany, which produces applications for calculation of fees for phone calls. The business needs an application that can calculate the total price for a call. A Call contains data like time of the call, duration and phone number. A call can also be a SMS or a data transfer. The Business states that it’s important that the algorithm is flexible with regards to current and future calculation models. That could be calculation of fee based pr. Second, pr. Minute, pr. Initiated second or initiated minute as well as different kinds of other models for campaigns. For data transfer calls they wish to calculate fees based on initiated KB or initiated MB of data transferred. There are also difference in price for call that cross country zones and calls that don’t.

Show extension points by use of interfaces, inheritance as well as other well-known software design principles and patterns

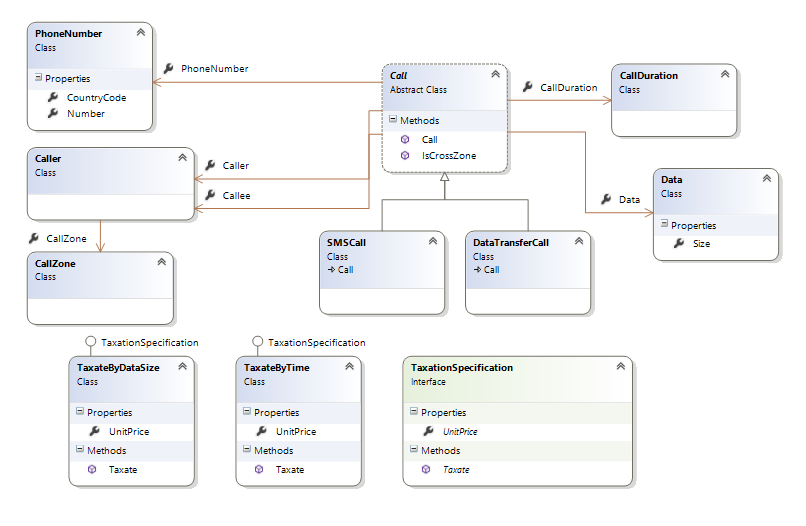
The Guy who is going to evaluate the solution has this statement on his LinkedIn profile.

**“Specialties: Solid, clean coder tackling complexity in the heart of software.”**

This clearly indicates that this guy is a fan of Robert Martin and Eric Evans. Luckily, I have read the two books Clean Code by Robert Martin and Domain Driven Design, tackling complexity in the heart of Software by Eric Evans. I am going to assume that the reader are familiar with the concepts used in these books. In order to speed up development I will make use of [Sharp Architecture](https://github.com/sharparchitecture/Sharp-Architecture) which is a library/framework facilitating Domain Driven Design. For instance I Get Entity and ValueObject base classes for free, so I don’t need to implement Equals operators and GetHashCode for my domain classes. This is very handy and fits well with NHibernate.

## Domain Layer

First I will make a quick Class diagram in order to list some candidates for concepts. This is just an initial overview and not the end result. I will use a test driven approach to flesh out the details of the design. This will undoubtedly lead to changes to the initial guess.

*Fig. Initial overview of domain concepts after brainstorming*

Some assumptions. Looking at the fact that we are going to create an application for calculation of fees for phone calls; lead me to think that the calls themselves has taken place already and has been saved as data in another bounded context.

Other thoughts. The fact that we can use various different ways of calculating the fee for a call leads the thoughts to the Strategy Design Pattern. We can also combine some of these strategies for instance taxate pr. Second plus extra fee for cross country zone calls. This calls for the Decorator Design pattern.

## Object Oriented Analysis and Design

Let’s continue the analysis by assigning responsibilities to objects or concepts.

1. Who or what should determine what taxation strategy should be used?
2. Who or what should control or initiate the calculation?
3. Who or what should contain the calls?

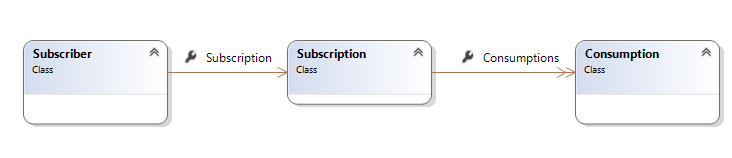
We want to debit some entity for the calls made. We haven’t yet mentioned a subscriber or customer. Let’s introduce a Subscriber and a subscription. It would make sense that calculation strategies and campaigns can be determined based on what subscription a subscriber has. The call itself will contain information about Cross Country Zone calls. So let’s assume that question 1 has been answered so far.

Regarding question 2. The only thing I can think of at the moment is some Taxation class or some manager class. One should always think twice about introducing such classes. Maybe there are other or better candidates. But we can change the design later if we discover some better solution.

Regarding question 3. I would say that the subscription should contain the calls.

After communication with a domain expert, I discovered that the Domain specific term for the common denominator for call, SMS and DataTransfer is **‘Consumption’**. The Domain specific term for an Algorithm or calculator is **‘Rating’**. I Will Refactor/Rename the code to reflect this.

We could imagine an object hierarchy where Subscriber has a subscription and the subscription has the list of consumption entries.



What we try to define is what aggregates we should have. Let’s take a closer look at the above 3 classes.

Subscriber is clearly an entity, since it has an identity and could be modified over time. Is Subscription an entity or a value object? Again I would say Entity. Subscription is either a reference to a static subscription shared by many subscribers having the same subscription type or it’s a new unique object owned uniquely by the subscriber. I will assume the last. Is Consumption an entity or a value object? It could be regarded as a value object in our bounded context since it’s immutable and can be distinguished by its values. There will not be two identical Consumption objects (Correct me if I’m wrong). If the domain requires it to be an entity we can change it later.

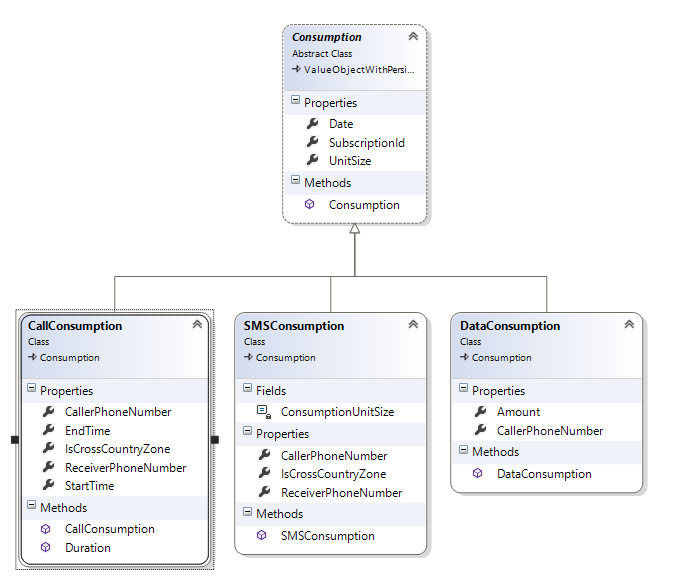
Vaughn Vernon suggests that Aggregates should be kept small. ([See Implementing Domain Driven Design](http://www.amazon.com/Implementing-Domain-Driven-Design-Vaughn-Vernon/dp/0321834577)).

Other Discoveries:

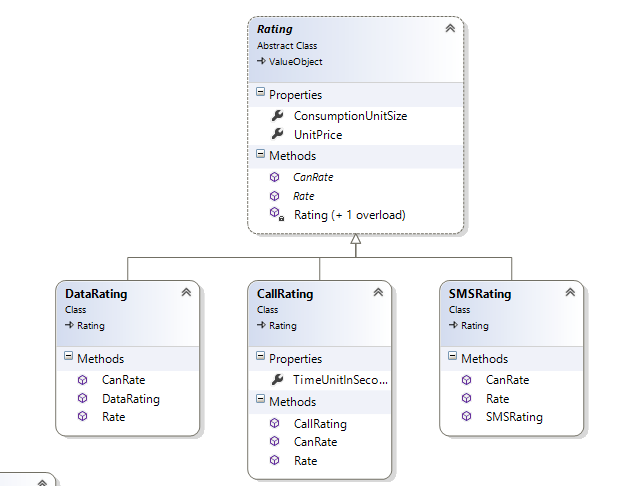
Initially I wanted to put the unitsize of a consumption on the Rating, but changed my mind, since the design was cleaner if unitSize is placed on the Consumption itself; This leads to a design where we in our bounded context doesn’t even care about unitsize, given that the consumption has stored the amount of units consumed. It leads to a discussion about who should be responsible for converting for example a timespan to an amount of ratable units. An important consideration is whether the unit sizes are fixed, o they need to be changeable.

I Added 3 properties to a Subscription one for each type of Rating, CallRating, SMSRating, DataRating. A Better way maybe would be to just have a list of ratings, and then in a more generic/polymorphic way choose what rating should rate a consumption. So I Added the bool CanRate(Consumption consumption) on the Rating abstract Class.

The final result for the domain can be seen in the following class diagrams.



*Fig. Consumption classes*



*Fig. Rating classes*

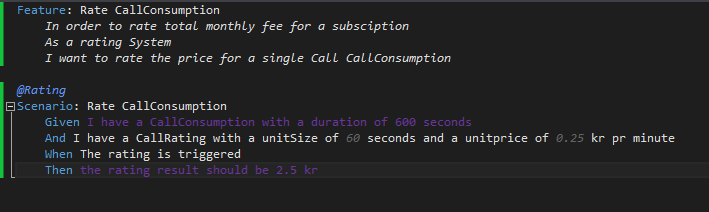
## 

*Fig. Subscription classes*

## Behavior Driven Development (BDD)

Let’s define some features and requirements to drive our tests. I could use xUnit directly or I could use Specflow either on a high level acceptance criteria way or in a more technical BBD level way.

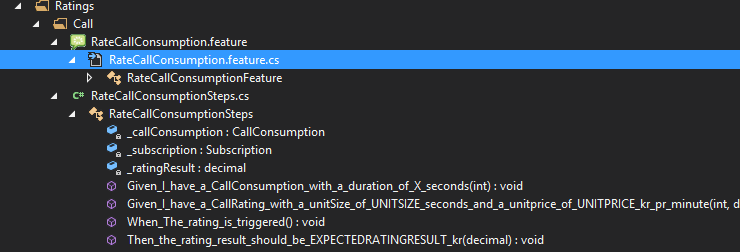
As an example let’s define the following feature.

*Fig. Some actions are purple even if they are bound to a method (Looks like a bug in SpecFlow)*

SpecFlow will generate a test skeleton from the above Gherkin definition.

You have to look in the Visual Studio Solution to see the code.

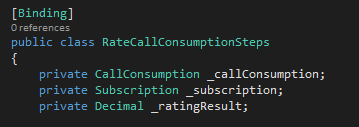
Below is a screenshot of the solution explorer giving an idea about the structure.



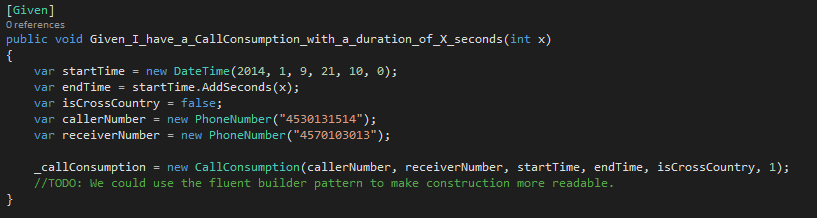
*Fig. Generated RateCallConsumptionSteps.*

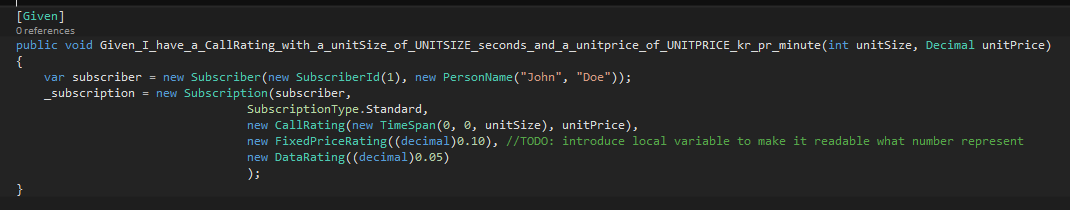
My job as a developer is to fill out the generated methods following a AAA (Arrange, Act, Assert aka. Given, When, Then) pattern.

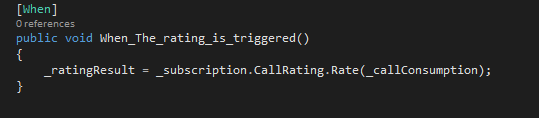
1. First I add private member variables



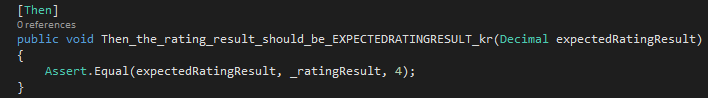
1. Add construction of a callConsumption object



1. Add construction of CallRating
2. Add implementation of the ‘When’ step



1. Add implementation of the ‘Then’ step.

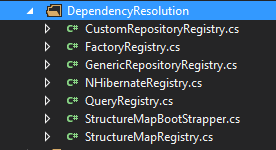


## Infrastructure

I’ve added the implementation of repositories to the MVNO.Infrastructure project. I have again used SharpArchitecture base classes in order not to re-invent everything. QuerySpecifications residing in the Domain layer can be passed to the repositories which will translate them to NHibernate queries.

## StructureMap

I have created several Registry classes which takes care of registering different types with the StructureMap ObjectFactory.



I only had time to finish some of them.

## Presentation Layer and making the whole website

Let’s move on to creating a UI prototype using ASP.NET MVC.

Let’s say we want to create a monthly overview of the consumption for a subscription.

We will group the Consumptions by type: Call, Sms and Data.

Display the total consumption for each group and display the corresponding rating.

I strongly believe in having a dedicated viewmodel for all data you want to display in the views.

I Created **MonthlyConsumptionViewModel** and **ConsumptionGroupViewModel**

I Also Created a **ConsumptionsController**

The ConsumptionController gets a Repository injected through the constructor. So StructureMap is working!

Ideally I would inject commands, queries or applicationService instances, rather than using repositories directly.

## Conclusion

Time is running out…

It’s my hope that my programming style and philosophy has been demonstrated. The main goal has been to qualify myself as a skilled programmer suitable for joining the MVNO team. I might have bend the scope of the task, which has led to a non-completed application, I hope that’s not a problem.

What have we accomplished? I have tried to reveal some of the thought works going on while trying to solve this task.

I have demonstrated

1. OOAD using a DDD way of thinking, trying to form a Ubiquitous language within a bounded context.
2. Used a lot of DDD concepts in praxis.
3. The use of SpecFlow and xUnit
4. Use of Hexagonal/Onion architecture where the Domain layer is isolated, independent and persistence ignorant.
5. Use of a 3rd party library ‘Sharp Architecture’ in order to speed things up.
6. Use of StructureMap
7. Use of ASP.NET MVC

What I haven’t demonstrated

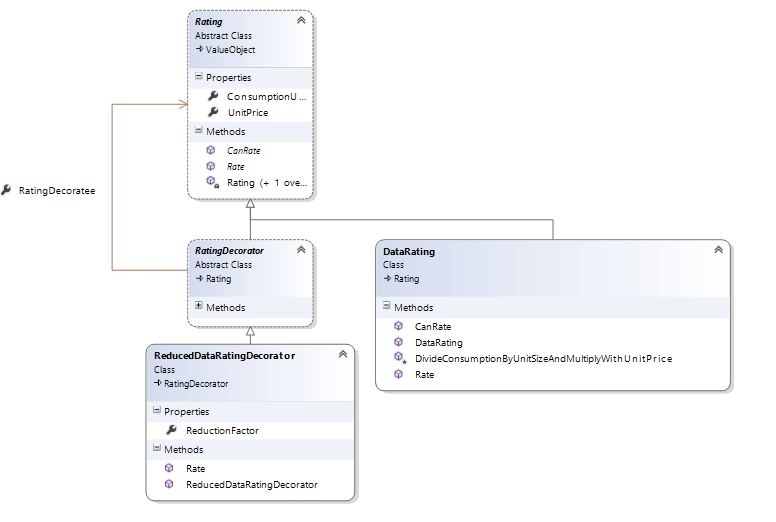
1. The whole application hasn’t been fully wired up.
2. SQL
3. A lot of asp.net MVC concepts like action filters, model binders, routing etc.

What could I have done differently?

1. I could have focused even more on SOLID and worked more with Strategy Pattern and decorator pattern.
2. I could have written more about overall architecture. Consuming Events from other bounded contexts for example. Or Batch jobs that Generate reports for the consumptions for each month for subscriptions.

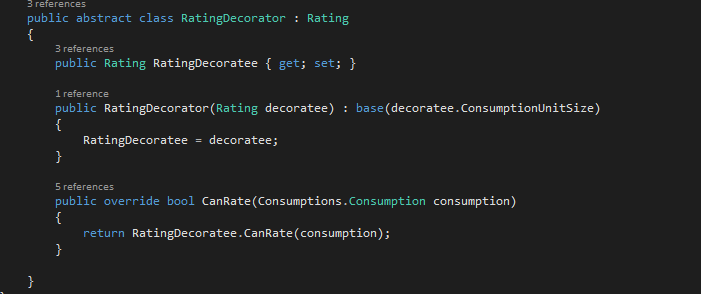
## Decorator Revisited

The following is a refactoring added after the task was delivered. I did mention Decorator pattern, but I never made an implementation. I will show this here.



*Fig. abstract RatingDecorator and Concrete ReducedDataRatingDecorator class.*

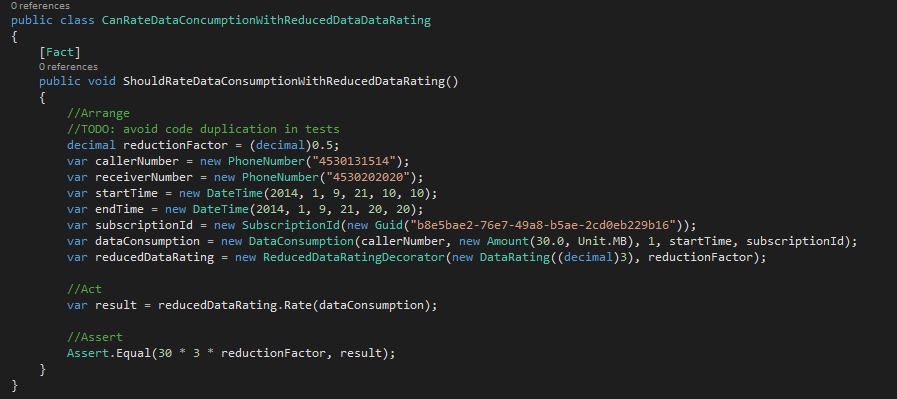
Assume we have one existing Rating class. And we want to do something extra with a specific type of consumptions. In this case we can use the decorator pattern to extend the behavior found on the existing Rating class. This class we name the ‘Decoree’. A Decorator needs a reference to the Decoratee and it needs to implement the same abstraction as the Decoratee. So we need a new abstract class that we will call RatingDecorator.



*Fig. The abstract RatingDecorator that inherits the Rating abstraction.*

Notice the property for the Rating Decoratee which can be injected through the constructor. One could argue that it should be a private setter on that property; and the parameter less constructor should be protected.

Let´s define a test that can assert that a new concrete DataRatingDecorator will work.

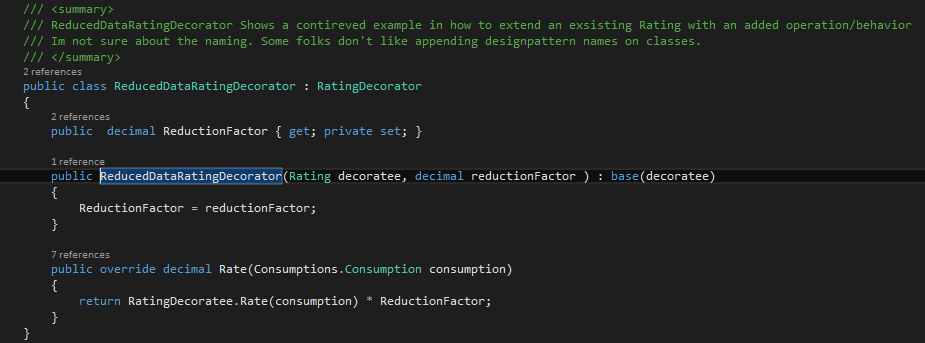


*Fig. new xUnit test for testing a concrete RatingDecorator.*

The xUnit test has a lot of duplicated setup code copied from another test. This should be moved to another class In order to obey the D in SOLID principles (Avoid code duplication).

Notice that a DataRating is new’ed and injected into the ReducedDataRatingDecorator constructor.

Now we need to do the implementation of the ReducedDataRatingDecorator.



*Fig. Implementation of ReducedDataRatingDecorator.*

There are no check of ReductionFactor, so theoretically an overflow could occur during the Rate operation. This is very little code required to extend the original Rate behavior with new Behavior. Another thing to be aware of is that the construction of the assembled rating behavior needs to be change as well. This code could reside in a factory class in order to avoid that change in construction strategy requires modification a lot of places in the code (We try to follow the O in SOLID (Open closed principle).