C#Corner

THE SOLID PRINCIPLES

Gaurav Kumar Arora

The SOLID Principles

Approach to Write Robust Programs

This free book is provided by courtesy of C# Corner and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. Please do not reproduce, republish, edit or copy this book.

Dedicated to my Younger sister - kanchan

Gaurav Kumar Arora

Author, C# Corner

Sam Hobbs

Editor, C# Corner

Table of Contents

Myths related to SOLID

There are many myths related to solid before going further lets first take a look about these myths and how much these myths are having real existence.

I also called these as a stupid excuses not to implement the SOLID things or by these some techies hide their main motive. Anyways, I do not want to go in this battle.

I found there are a few myths related to SOLID that cause architecture and Design Patterns to be in a confusing mess.

There are many kinds of myths, some related to the structuring of programs and others related to Design Patterns.

I know OOP so why do I need to learn S.O.L.I.D?

This is the biggest mistake when someone relates S.O.L.I.D to Object Oriented Programming. Earlier, I was one of these people who believed that myth. OOP is a programming paradigm based on concepts of object whereas S.O.L.I.D. are the programing principles that tell us how to write good programs.

I know Design Patterns so why do I need to learn S.O.L.I.D?

Again a mistake, design patterns are telling how to design our programs/software but on the other hand S.O.L.I.D are only principles to make our program clean.

SOLID Principles are only applicable to .NET/C# and not for Java

One great misunderstanding about these Principles a myth, which says these principles are not applicable for Java. Ah! What a myth :). These Principles are not related to any programming language or in other words these principles are not built for any specific programming languages. These principles are just kind of guidelines to make our code/program robust and it does not matter in which language the program has been written.

As an Architect, do I really need to care about S.O.L.I.D?

A myth talked by Mr. Nangal one of our readers, I am taking the honor to include the same. It is purely un-realistic if someone say that these principles are not for Architects. As an architect - you should think about a robust design, scalability, and components distribution.

While designing any software/application, one should keep in mind all these principles. I agree there would chances for overlapping but to make application design robust one should follow these.

These principles are applicable for all who involved in Technical part of the Software.

A Developer - needs to write a code in way so, he/she must obey these principles

A Reviewer - should review code/area in a way so, he/she must obey these principles

A Tech Lead - should guide the team in a way so, team must obey these principles

There may be long list depending upon the nature and size of a software/application.

I am working in maintenance project, why do I care about S.O.L.I.D?

There may be long list depending upon the nature and size of a software/application.

Interesting one, it vary project to project how one could implement these principles. But, we can't say that in a maintenance type of project we neglect these principles.

I would take the opportunity to elaborate this with a real scenario example: a long time back, I worked on a maintenance project and the interesting thing is that when I get a part of that team, the project was almost done. I got some assignment and I noticed a much repeated code through-out the application. I have to complete that assignment in 16Hrs (2days) including QA's efforts.

I approached to my Team Manager, unfortunately my manager wasn't convince from me, his words was "we are almost done with this project, I do not care about the SOLID principles, also, we have received a Green flag from our users/clients. If you can't complete this task within time limit then I have to assign this task to someone else, choice is yours“.

It wasn't my day that day. Can you imagine, what did I do that time?

Many of us will definitely complete the task within a stipulated time period. We are developers and we don't want to leave tasks.

Of course, I did not leave that task, I completed the same before time period but in my way of style. I wrapped up a new class and added new functionality related to my task, wrote it in a way of S.O.L.I.D. also, attached the similar things with this class so, other area of code could feel S.O.L.I.D. When I sent my changes for review, I sent with these notes:

"I noticed lot of code is repeating itself and some places are yak, need to clean up the things. I implemented my changes by obeying S.O.L.I.D. Principles".

Can you imagine what would be the reply of code review?

Ah! I got a reply from a reviewer, who was much senior to me quoting notes "Gaurav, I am reviewing this project from last 17-months and did not find any discrepancies. However, I noticed few things in your code as mentioned below, request you to make appropriate changes."

I made all the requested changes, which are just cosmetic things not related to S.O.L.I.D or my way of coding and my changes get approved after QA and deployed with release.

Now, think the story about this incident, we got a call for code-review by the client. We have a meeting, I wasn't the part of that meeting only few senior guys of our team, attended.

Suddenly, I got a phone call on my desk from my manager with words "Come at conference hall, client is calling". It was a video conference call, and suddenly my manager introduced me to client.

I was shocked when client asked about my work experience and the approach why I should write the code in this way. Crap man! I was not in my zone that time.

Suddenly client clapped and said 'great, you're done it in a good manner'. ...

My motto to give the above scenario is just, what if I felt that I am a developer and working in a maintenance project also, my project is having final release, why should i bother about all these principle.

As, I said, it depends/vary project-to-project that how to implement these principles, but it is recommendable never forget these principles.

Introduction

Object Oriented Programming (OOP) PROVIDES a way to write and polish our programs in a better shape and way. These are basic points that guide us for how to write a good program.

For great Object Oriented Designs (OOD) we need to think beyond this. S.O.L.I.D principles provide us a way to write great design, yes there are certain design patterns, that guide us the same but SOLID exist much before these patterns

Let's discuss and understand all of that using C# code snippets as in the following:

Defining S.O.L.I.D

This is an acronym introduced by Michael Feathers as:

S for **SRP**: Single Responsibility Principle.

O for **OCP**: Open/Closed Principle.

L for **LSP**: Liskov Substitution Principle.

I for ISP: Interface Segregation Principle.

D for **DIP**: Dependency Inversion Principle.

Briefing Single responsibility principle (SRP)

Name of this principle described itself, it should have single responsibility. Who should have single responsibility? Here, we are studying/learning principles to design best classes/programs/systems.

In reference to this I would say "A class should have single responsibility". Let's dive into ocean – can we read this like "a class should not design to do multiple activities", so, what kind of activities

Let's think I have to design a system which provide me employee details, so, it should include activities, general I have CRUD (Create Read Update Delete) operations. Now, as per Single responsibility principle, I have to design a class, which should do any of these operations but not all of these :)

I am remembering my old days, when I was learning C++. Generally, I was writing 1000s lines of code in a one program contains many if…else. At that time I was happy to run these programs.

Now, today's I do not like a method which contains more than 4-5 lines, how world changed?

When I was learning this principle, question was in my mind, why class should not responsible for multiple responsibility? I found the answer as:

More, responsibility tied classes towards more changes in future.

Yes, this is true, if I designed a class (I will explain using my old days code in coming learning part of S.O.L.I.D), which is responsible to modify data, retrieve data and then save data. In future if there is such kind of business requirements, where our modification or data retrieval logic would be changed then we have to change our classes many time and at many places, this would be encountered more bugs and more code changes.

Finally, we can define Single responsibility principle as:

"*Class should be designed for single responsibility and there should not more than one reasons to make changes in this class. The responsibility of this class should be completely tide/encapsulated by the class*."

Learning Open/closed principle (OCP)

When, I read this principle, I thought it looks like that my class should be open or closed, I thought either one. But when I read following definition from wiki:

"software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"

I was shocked to think how it is possible to make my class open and closed and not open or close, I thought in other words how can I allow things to modify without doing actual modifications to my object, it was just confusing to me

I dive into OOPs (Object Oriented Programming) for answer to my question. Let's think about abstraction; we can create a base class and veritable functions with different behavior. Yes, we can allow changes by keeping objects unchanged.

Let take an example: We have to send an email for different operations body of emails depend upon certain business rules and can contain different or same messages. Now, what we can do here, we can create a class like CreateEmail or whatever you want to name it, with one method BuildMessage. So, this class is only and only responsible to build email messages as per different logics as this method is over-

riddable I can define it functionality per my choice. Isn't it interesting and easy [In coming parts, I will try to explain in details using code-examples].

Learning Liskov substitution principle (LSP)

Here is definition from wiki:

"*if S is a subtype of T, then objects of type T may be replaced with objects of type S (i.e., objects of type S may be substituted for objects of type T) without altering any of the desirable properties of that program (correctness, task performed, etc.)*"

I understood above definition like this: parent should easily replace the child object.

To understand it bit more, let's look into example of EmailNotifications(considered in OCP, above), now let's say we need to also send this email for print what we can do? We can create a new class, let's call it NotificationsForPrint or whatever name you would like :) it inherits our class EmailNotifications. Now, out both classes base and child class having at least one similar methods.

Can we use our child class to substitute our base class, no, in this situation never? So, we need to use inheritance, define two separate interfaces one is building message and another one is sending message and let's decide in implementation where for what we need to build and send messages. I will explain this example in details – in coming parts.

Learning Interface segregation principle (ISP)

Here is a definition from wiki:

"*No clients should be forced to implement methods which it does not use and the contract should be broken into small and more specific interfaces.*"

I took this correct as: "as a client why should I implement 9-methods of interface when I need only 3-methods", isn't it make developers life easy

This also similar to High Cohesion Principle of GRASP.

Think, can we consider our EmailNotification example (for both scenarios print and send via smtp server)?

Learning Dependency inversion principle (DIP)

This principle remembers us Decoupling

"*High level modules should not depends upon low-level modules both should be tide using abstractions*"

What does this mean? Let's take a look into our **EmailNotification** example once again and think why should our code decide where to send my email (smtp server or printer) at very beginning.

Single Responsibility Principle (SRP)

The principle is self describing, there should be a single responsibility. Who should have a single responsibility? Here, we are studying/learning principles to design the best classes/programs/systems.

In reference to this I would say "*A class should have a single responsibility*". Let's dive into the ocean. To read this like "a class should not be designed to do multiple activities", so, what kind of activities?

Let's think. I need to design a system that provides me employee details, so it should include activities. Generally I have Create, Read, Update and Delete (CRUD) operations. Now, as in the Single responsibility principle, I need to design a class, which should do any of these operations but not all of these.

"*I am remembering my old days, when I was learning C++. Generally, I was writing 1000s lines of code in a one program containing many if..else. At that time I was happy to run these programs*".

Now, today I do not like a method that contains more than 4-5 lines, how has the world changed?"

When I was learning this principle, the question was in my mind, why is a class not responsible for multiple responsibilities? I found the answer as:

It is more a matter of the responsibility of classes being tied towards more changes in the future.

Yes, this is true, if I designed a class (I will explain using my old days code in a future learning part of S.O.L.I.D), that is responsible to modify data, retrieve data and then save data in the future if there is.

For such kinds of business requirements, where our modification or data retrieval logic would be changed then we need to change our classes many times and in many places, this would introduce more bugs and more code changes.

Now, have a look at the following snippet:

```
01. public class DataMigrater
02. {
03.     public IList<ServerData> GetData(DateTime initialDate, DateTime endDate)
04.     {
05.         //get server data within date range
06.
07.         return new List<ServerData>();
08.     }
09.
10.     public IList<ServerData>ProcessData(IEnumerable<ServerData>rawData)
11.     {
12.         //apply rules
13.
14.         return rawData.ToList();
15.     }
16.
17.     public void Migrate(IEnumerable<ServerData> rawData)
18.     {
19.         //migrate processedData from server to server
20.     }
21. }
22.
23. class Program
24. {
25.     private static readonly DateTime StartDate = new DateTime(2014, 07, 01);
26.     private static readonly DateTime EndDate = DateTime.Now;
27.
28.     static void Main(string[] args)
29.     {
30.         var dataMigrater = new DataMigrater();
31.
32.         //get raw data
33.         var rawData = dataMigrater.GetData(StartDate, EndDate);
34.
35.         //process raw data
36.         var processedData = dataMigrater.ProcessData(rawData);
37.
38.         //finally migrate processed data
39.         dataMigrater.Migrate(processedData);
40.
41.     }
42. }
```

In our class **DataMigrater** has too many responsibilities. This class:

- Fetches data
- Processes data
- Then, migrates the data

So, what is the SRP that our class is doing wrong? Here, we are not following S.O.L.I.D. What should our class do? Let's start from the name of the class, in other words **DataMigrate**, this looks to me to be a class that should be responsible only for migrating data. So, the class should not be concerned with what and how the data is coming for migration.

One more reason, a class should be responsible for only one thing. Let's think of a scenario where the method:

```
01. public void Migrate(IEnumerable<ServerData> rawData)
02. {
03.   //Stuff goes here
04. }
```

Throws an exception and our data does not get migrated. Now, we need to verify all three methods because we are not sure if the data is well fetched or well processed. This has provided us many burdens.

Do you want to bear this load? I am sure, no developer wants to bear this.

Now, a big question is, how to do that? Have a look at the following snippet:

```
01. public class DataMigrater
02. {
03.     private readonly IList<ServerData> _data;
04.     private readonly IServerDataRepository _repository;
05.     private readonly ILog _logger = LogManager.GetLogger(typeof(DataMigrater));
06.
07.     public DataMigrater(IList<ServerData> data, IServerDataRepository repository)
08.     {
09.         _data = data;
10.         _repository = repository;
11.     }
12.
13.     public void Migrate()
14.     {
15.         try
16.         {
17.             foreach (var data in _data)
18.             {
19.                 var stopWatch = Stopwatch.StartNew();
20.
21.                 Migrate(data);
22.
23.                 stopWatch.Stop();
24.
25.
_logger.InfoFormat("Total data {0} migrated in {1}", _data.Count, stopWatch.Elapsed);
26.             }
27.         }
28.         catch (Exception ex)
29.         {
30.             _logger.Error(ex);
31.
32.             throw;
33.         }
34.
35.     }
36.     private void Migrate(ServerData data)
37.     {
38.         try
39.         {
40.             //Migrate data from server to server
41.
42.             _logger.InfoFormat("Migrating data with Id:{0}", data.Id);
43.         }
44.         catch (Exception ex)
45.         {
46.             _logger.ErrorFormat("An exception occurred attempting to migrate data with Id:
{0}", data.Id);
47.
48.                 throw;
49.         }
50.     }
51.
52. }
```

Now, our class has only one method, **Migrate()**. You noticed that there are many changes made in this class, we will discuss all one-by-one in future articles. As of now let's concentrate on the Single Responsibility Principle.

In the preceding, our class is now only concerned with the Migrate Server data. This class is not concerned with whether the supplied data is processed or raw, there are other classes responsible for these things now.

See the following:

```
01.  public class ServerProcessedOrRawDataQuery
02.  {
03.      internal IServerDataRepository Repository { get; set; }
04.      public ServerProcessedOrRawDataQuery(IServerDataRepository repository)
05.      {
06.          Repository = repository;
07.      }
08.      public IQueryable<ServerData> Query(DateTime startDate, DateTime endDate)
09.      {
10.          return new ServerDataQuery(Repository).ProcessedData()
11.              .Where(d => d.InitialDate <= endDate && d.EndDate >= startDate);
12.      }
13.  }
14.  ublic class ServerDataQuery
15.  {
16.      internal IServerDataRepository Repository { get; set; }
17.
18.      public ServerDataQuery(IServerDataRepository repository)
19.      {
20.          Repository = repository;
21.      }
22.      public IQueryable<ServerData> Query()
23.      {
24.          return Repository.Get().AsQueryable<ServerData>();
25.      }
26.
27.      public IQueryable<ServerData> ProcessedData()
28.      {
29.          return Query().Where(d => d.IsDirty == false);
30.      }
31.
32.  }
```

Now, our program is changed as:

```
class Program
{
    private static readonly DateTime StartDate = new DateTime(2014, 07, 01);
    private static readonly DateTime EndDate = DateTime.Now;

    static void Main(string[] args)
    {
        var repository = new ServerDataRepository(); //we can use any DI to initiate our reposit

        var processedData = GetProcessedData(repository);

        var dataMigrater = new DataMigrater(processedData,repository);

        Console.WriteLine("Data migration started...");

        dataMigrater.Migrate();

        Console.WriteLine("Data migration completed...");

        Console.ReadLine();

    }

    private static IList<ServerData> GetProcessedData(IServerDataRepository repository)
    {
        return new ServerProcessedOrRawDataQuery(repository).Query(StartDate, EndDate).ToList();
    }
}
```

Finally, we can define the Single responsibility principle as:

"*A class should be designed for single responsibility and there should not be more than one reason to make changes to this class. The responsibility of this class should be completely tied/encapsulated by the class*."

Open/Closed Principle (OCP)

When I first read this principle, I thought it meant my class should be open or closed, I thought either one. But when I read the following definition from the wiki:

"Software entities (classes, modules, functions, and so on) should be open for extension, but closed for modification"

I was shocked to think how it is possible to make my class open and closed and not open or closed. In other words, I thought how can I allow things to modify without doing the actual modifications to my object, it was just confusing to me.

I dive into Object Oriented Programming (OOP) for answer to my question. Let's think about abstraction; we can create a base class and overridable functions with different behavior. Yes, we can allow changes by keeping objects unchanged.

Let's take an example: We need to send an email for various operations in the body of emails depending upon certain business rules and can contain different or the same messages. Now, what do we need to do here?

We can create a class like CreateEmail or whatever you want to name it, with one method BuildMessage. So, this class is only responsible for building email messages as per different logic. Since this method is overridable I can define its functionality as I choose to.

Doesn't it look very interesting and easy?

In continuation with our preceding code of example, we need to update our database from one server to another server (suppose we need to refresh our Development database from production), but there are some rules, like the data-type should be the same, the data value should be changed and so on.

```
01. public class ValidateData
02. {
03.     public void SyncronizeData(ServerData data, SourceServerData sourceData)
04.     {
05.         if (IsValid(data, sourceData))
06.         {
07.             //save data
08.         }
09.     }
10.
11.     private bool IsValid(ServerData data, SourceServerData sourceData)
12.     {
13.         var result = false;
14.
15.         if (data.Type == sourceData.Type)
16.             result = true;
17.
18.         if (data.IP != sourceData.IP)
19.             result = true;
20.
21.         //other checks/rules to validate incoming data
22.
23.         return result;
24.     }
25.
26. }
```

Can you think, what is wrong with the preceding code?

Let's discuss now, have a look at the class ValidateData. First of all it is doing two activities, in other words our class is responsible for the following two things:

- To validate incoming data (from source server)
- To save data

Now, think of the scenario that someone wants to extend this, so it could use another external service. In this scenario he/she would have no other choice but to modify the **IsValid** method. Also, if someone must make it a component and provide it to third parties for their use, then its users would have no way to add another service. This means this class is not open for extensions. On the other hand if someone needs to modify the behavior to persist data, they need to change the actual class.

To sum-up, this class is directly violating OCP since this is neither open for extensions nor closed for modifications.

So, what would be a better solution for this class, so it conforms to OCP?

Remember abstraction, let's try to do something by creating an interface:

```
01. public interface IDataValidator
02. {
03.     bool Validate(ServerData data, SourceServerData sourceData);
04. }
05.
06. public class IPValidator : IDataValidator
07. {
08.     public bool Validate(ServerData data, SourceServerData sourceData)
09.     {
10.         return data.IP != sourceData.IP;
11.     }
12.
13. }
14. public class TypeValidator : IDataValidator
15. {
16.     public bool Validate(ServerData data, SourceServerData sourceData)
17.     {
18.         return data.Type == sourceData.Type;
19.     }
20. }
```

The preceding code snippet is self-explanatory, where we defined **IDataValidator** that has a method Validate.

The method name describes itself, it's a part of **DataValidator**, and so it should validate data, so it is named Validate.

Now, redesign our class **ValidateData**:

```
public class ValidateData
{
    public bool IsDataValidated(ServerData data, SourceServerData sourceData)
    {
        IList<IDataValidator> validators = new List<IDataValidator>();
        validators.Add(new IPValidator());
        validators.Add(new TypeValidator());

        return IsDataValid(validators, data, sourceData);
    }


  private bool IsDataValid(IList<IDataValidator> validators, ServerData data, SourceServerData
    {
        foreach (var validator in validators)
        {
            if (validator.Validate(data, sourceData))
            return true;
        }
        return false;
    }
}
```

Stay here to discuss the preceding snippet! In the preceding we have a ValidateData class that is only responsible for validating data by certain validations/rules.

With the preceding changes, our class is not more stable and robust, we can add as many validators as we want. Also, you can use this validator to save your data.

Ah! I forget to mention, you can save the data just by calling this validator from another class, it could be a repository class or your custom class where you just persist your data.

I am not going to write that part of save, you can easily implement this by yourself.

Liskov substitution principle (LSP)

Let’s understand this principle from following definition of wiki:

"*if S is a subtype of T, then objects of type T may be replaced with objects of type S (in other words, objects of type S may be substituted for objects of type T) without altering any of the desirable properties of that program (correctness, task performed, and so on)*"

I understood the preceding definition to bean that the parent should easily replace the child object.

To understand it a bit more, let's look into an example of **EmailNotifications** (considered in OCP, above). Now let's say we need to also send this email for printing, what can we do?

We can create a new class. Let's call it **NotificationsForPrint** or whatever name you would like. It inherits our class **EmailNotifications**. Now, both classes, the base and child classes, have at least one similar method.

Can we use our child class to substitute our base class? No, in this situation, never. So, we need to use inheritance. Define two separate interfaces, one for building the message and another for sending the message and let's decide on the implementation of where and for what we need to build and send messages.

Have a look at the following snippet:

```
public class DataBase
{
    public virtual bool IsValid(ServerData data, SourceServerData sourceData)
    {
        return new Validator(new List<IValidator>()).Validate(data, sourceData);
    }

    public virtual void Save()
    {
        //logic to save data
    }
}

public class ProdDB : DataBase
{
    public override bool IsValid(ServerData data, SourceServerData sourceData)
    {
        return base.IsValid(data, sourceData);
    }

    public override void Save()
    {
        //logic to save data
        base.Save();
    }
}

public class QADB : DataBase
{
    public override bool IsValid(ServerData data, SourceServerData sourceData)
    {
        return base.IsValid(data, sourceData);
    }

    public override void Save()
    {
        //logic to save data
        base.Save();
    }
}

public class LocalDB : DataBase
{
    public override bool IsValid(ServerData data, SourceServerData sourceData)
    {
        return base.IsValid(data, sourceData);
    }

    public override void Save()
    {
        throw new Exception("Local Data should not be saved!");
    }
```

Recall inheritance and you can visualize that DataBase is a parent class of ProdDB, QADB and LocalDB.

Let's think of polymorphism for a while and we can write it as in the following

```
DataBase pDataBase = new ProdDB();
DataBase qDataBase = new QADB();
DataBase lDataBase = new LocalDB();

//also can create a list of DataBase type

var dataBases = new List<DataBase> {new ProdDB(), new QADB(), new LocalDB()};
```

Isn't it easy to save my object using this?

```
var dataBases = new List<DataBase> {new ProdDB(), new QADB(), new LocalDB()};
foreach (var dataBase in dataBases)
{
    if (dataBase.IsValid(data, sourceData))
        dataBase.Save();
}
```

What's wrong in the preceding, nothing?

Yes, you are absolutely correct, there is nothing wrong with the preceding code, the only thing is, its execution. When the preceding code executes, it will also invoke a save method of the LocalDB object. In this case we received an exception since our LocalDB object is not supposed to save data.

A big question is, why did this happen?

In simple words LocalDB is actually not an entity of DataBase or we can say DataBase is not an actual parent of LocalDB.

Another question is, why is LocalDB not an entity of DataBase, when it inherits DataBase?

Hold on, go back to the LocalDB class and check, This is not meant to implement a Save() method, here this makes LocalDB a separate entity.

In simple words, LISCOV says the parent should easily replace its child.

How to implement LISCOV principle?

We know LocalDB is not supposed to save data but others are. Let's consider the following snippet:

```
01. public interface IRule
02. {
03.     bool IsValid(ServerData data, SourceServerData sourceData);
04. }
05.
06. public interface IRepository
07. {
08.     void Save();
09. }
```

We now have two interfaces with their own methods. IRule to validate data and IRepository to save/persist data.

Let's make changes to our LocalDB class, as:

```
01. public class LocalDB : IRule
02. {
03.     public bool IsValid(ServerData data, SourceServerData sourceData)
04.     {
05.         return new Validator(new List<IValidator>()).Validate(data, sourceData);
06.     }
07. }
```

Why did we implement IRule?

For LocalDB, we only want to check whether the data is valid or not. We do not want to persist the data in any scenario.

Now, we can't write this:

```
DataBase lDataBase = new LocalDB();
```

Our DataBase class, should be like this:

```
public class DataBase : IRule, IRepository
{
    public virtual bool IsValid(ServerData data, SourceServerData sourceData)
    {
        return new Validator(new List<IValidator>()).Validate(data, sourceData);
    }

    public virtual void Save()
    {
        //logic to save data
    }
}
```

The other classes will remain unchanged.

Our execution code goes as

```
public void Execute(ServerData data, SourceServerData sourceData)
{
    var dataBases = new List<IRepository> { new ProdDB(), new QADB() };

    foreach (var dataBase in dataBases.Where(dataBase => ((IRule)dataBase).IsValid(data, so
    {
        dataBase.Save();
    }
}
```

Now, our code looks easier to handle.

Interface segregation principle (ISP)

Let’s understand this principle from following definition of wiki:

"*No clients should be forced to implement methods that it does not use and the contract should be broken into small and more specific interfaces.*"

I took this correct as: "as a client, why should I implement 9 methods of an interface when I need only 3 methods", doesn't it make a developers life easy?

This also similar to the High Cohesion Principle of GRASP.

Think, can we consider our **EmailNotification** example (for both scenarios print and send via SMTP server)?

Let's explore this with an example.

First of all go back and have a look at the code example discussed in LSP, there some of our databases are being saved after validation. Now, think of a scenario were there are more databases and for these additional databases we require a report in, other words new databases need to be read and saved.

```
01. public interface IRepository
02. {
03.     void Save();
04.     void Generate();
05. }
```

In the very first instance, I can think to add a new method to the interface **IRepository** (that can read data or generate a report).

Do you think the preceding approach is good?

Think, think and again think.... :)

By adding a new method to an existing interface, we are forcing the use of a new method to all those classes that are already implementing this interface. But those classes are not supposed to use any newly added method. So, my ProdDB class looks like:

```
01. public class ProdDB : DataBase
02. {
03.     public override bool IsValid(ServerData data, SourceServerData sourceData)
04.     {
05.         return base.IsValid(data, sourceData);
06.     }
07. public override void Save()
08.     {
09.         //logic to save data
10.         base.Save();
11.     }
12. public override void Generate()
13.     {
14.         //Report generation logic
15.     }
16. }
```

But actually, the **ProdDB** class does not require to Generate a report, but with the preceding implementation this class must be implemented in a new **Generate()** method.

Here, we are forcing our class to implement that method, that this class does not want.

So, we are not following the Interface Segregation Principle in our preceding code (go the top and read the ISP definition :)).

What is the solution for this problem?

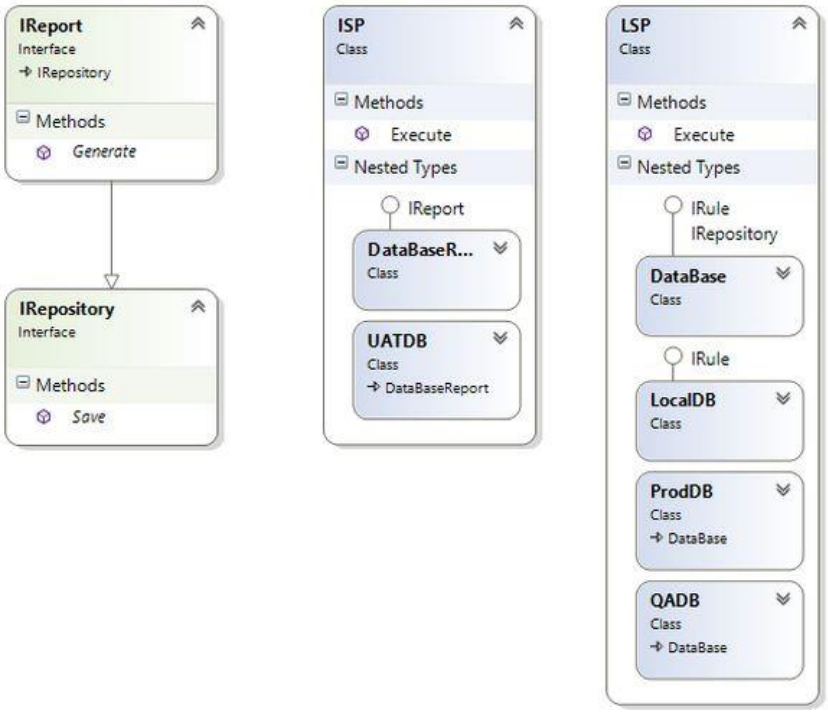
First, try to segregate our IRepository interface.

```
01. public interface IReport:IRepository
02. {
03.     void Generate();
04. }
```

To segregate, I created another IReport interface with the new method Generate(). Now, we have two separate interfaces, IRepository and IReport.

Let's create a new class that is meant for the clients that want to generate the report:

```
01. public class DataBaseReport : IReport
02. {
03.     public void Save()
04.     {
05.         var dataBase = new DataBase();
06.         dataBase.Save(); // we want to save data
07.     }
08.
09.     public void Generate()
10.     {
11.         //implement report generation logic here
12.     }
13. }
```



At this point, we have two different classes, **DataBase** and **DataBaseReport**. One is for those clients who don't want to generate a report and another is for those that do want to generate a report :)

So, our execute method would look like:

```
01. public void Execute()
02. {
03.     //Old client implementation
04.     IRepository repository = new DataBase();
05.     repository.Save();
06.     //implementation for new clients who want to generate report
07.     IReport report = new DataBaseReport();
08.     report.Generate();
09. }
```

You can see how we resolved the problem. This solution will be very useful when millions of clients need different things whereas our existing clients don't.

Dependency inversion principle (DIP)

Dependency inversion principle is something related to Decoupling.

Which says "*High level modules should not depend upon low-level modules, both should be tied using abstractions*"

What does this mean? Let's have a look into our **EmailNotification** example once again and think, why our code should decide where to send my email (SMTP server or printer) at the very beginning.

Why not it should automatically perform a preferable action (we will have a look at the details in future parts)? Until then have a look into this pattern on the wiki.

Let's explore this with an example.

In the Rewind code example, we discussed the Single Responsible Principle. There is a property Type, that tells the data type, now have a look at the following snippet.

```
01. public class DataBase : IRule, IRepository
02. {
03.     private ISession session;
04.     public virtual bool IsValid(ServerData data, SourceServerData sourceData)
05.     {
06.         //implementation goes here
07.     }
08.     public virtual void Save(ServerData data)
09.     {
10.         if(data.Type == 1)
11.         {
12.             session = new ServerDataSession();
13.         }
14.         else
15.         {
16.             session = new SourceServerDataSession();
17.         }
18.
19.         session.Save(data);
20.     }
21. }
```

There is a violation of SRP in the preceding, to solve this let's create a separate interface that holds a save method as in the following:

```
interface ISession
{
    void Save<T>(T data);
}
and lets create different saver as per following snippet:
class ServerDataSession:ISession
{
    public void Save<T>(T data)
    {
        //save logic goes here
    }
}
class SourceServerDataSession:ISession
{
    public void Save<T>(T data)
    {
        //save logic goes here
    }
}
```

Now what?

We need to supply a Session depending on Data type. Here we are delegating the responsibility to someone else. In other words, for the preceding snippet our DataBase class is delegating its responsibility to others (ServerDataSession and SourceServerDataSession). We need to modify our DataBase class as in the following:

```
public class DataBase : IRule, IRepository
{
    private ISession _session;
    public DataBase(ISession session)
    {
        _session = session;
    }
    public virtual bool IsValid(ServerData data, SourceServerData sourceData)
    {
        //implementation goes here
    }

    public virtual void Save(ServerData data)
    {
        _session.Save<ServerData>(data);
    }
}
```

At this point, our client is free to inject what he/she wants to consume:

```
IRepository dataBase = new Database(new ServerDataSession());
```

"*No clients should be forced to implement methods that it does not use and the contract should be broken into small and more specific interfaces*."

I took this correct as: "as a client, why should I implement 9 methods of an interface when I need only 3 methods", doesn't it make a developers life easy?

References

References used in this books:

- http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- http://en.wikipedia.org/w/index.php?title=Michael_Feathers&action=edit&redlink=1
- http://en.wikipedia.org/wiki/Open/closed_principle
- http://en.wikipedia.org/wiki/Liskov_substitution_principle
- http://en.wikipedia.org/wiki/GRASP_(object-oriented_design)
- http://en.wikipedia.org/wiki/Coupling_(computer_programming)
- https://github.com/garora/somestuff/tree/master/LearningSolid