|  |  |  |  |
| --- | --- | --- | --- |
| *.Net Version Release history* | | | |
| .Net Version | **Release Date** | **Tool** | **Feature** |
| 1.0 | 2002 | Visual Studio .Net | First release of .net  CLR 1.0 |
| 1.1 | 2003 | Visual Studio 2003 | Support for ASP.Net mobile controls Supports side-by-side execution [Security Changes](http://msdn.microsoft.com/en-us/library/h88tthh0(v=vs.90).aspx)  **CLR 1.1** |
| 2.0 | 2005 | Visual Studio 2005 | Generics (with generic collection) Nullable Types Support of IPv6 addresses in .net remoting **Common Language Runtime 2.0** |
| 3.0 | 2006 | – | WCF (Communication framework) WPF (Presentation framework) WF (Workflow Foundation) |
| 3.5 | 2008 | Visual Studio 2008 | LINQ Addin / Plugin Model (System.AddIn.Contract.dll) |
| 4.0 | 2010 | Visual Studio 2010 | Parallel Computing Code Contracts **Lazy Initialization** **Dynamic Language Runtime** In-process side-by-side hosting **Background garbage collection** Covariance and Contravariance **Common Language Runtime 4.0** |
| 4.5 | 2012 | Visual Studio 2012 | Enhanced regular expression support Default culture for application domain Zip compression Support of array with size more than 2GB Asynchronous file operation Improvement in parallel computing  **ASP.NET**  Support for new HTML5 form types.  Support for model binders in Web Forms. These let you bind data controls directly to data-access methods, and automatically convert user input to and from .NET Framework data types.  Support for unobtrusive JavaScript in client-side validation scripts.  Improved handling of client script through bundling and minification for improved page performance.  Integrated encoding routines from the Anti-XSS library (previously an external library) to protect from cross-site scripting attacks. |
| 4.5.1 | 2013 | Visual Studio 2013 | Ability to collect diagnostics information Ability to explicitly compact the large object heap (LOH) during garbage collection Additional performance improvements such as ASP.NET app suspension Multi-core JIT improvements |
| 4.6 | 2015 | Visual studio 2015 |  |
| 4.6.1 |  |  |  |
| 4.6.2 |  |  |  |
|  |  |  |  |

Note:  
- .Net 3.5, 3.0 and 2.0 uses same common language runtime version **2.0**.  
- .Net 4.5, 4.5.1 and 4.0 share same common language runtime version **4.0**.  
- I have not included language features in above list e.g. Partial classes / anonymous method. I will cover them in following section.

|  |  |  |  |
| --- | --- | --- | --- |
| *C# Language Release history* | | | |
| C# Version | **Release Date** | **Tool** | **Feature** |
| 1.0 | 2002 | Visual Studio .Net | First release of .net |
| 2.0 | 2005 | Visual Studio 2005 | Partial classes Support for generics Iterators Nullable syntax Anonymous methods Static class Volatile keyword |
| 3.0 | 2008 | Visual Studio 2008 | Implicitly Typed Local Variables Extension Methods Lambda Expressions Type Inference Object and Collection Initializers Anonymous Types Automatically Implemented Properties Expression Trees |
| 4.0 | 2010 | Visual Studio 2010 | Support for Covariance and Contravariance Optional parameters and named arguments Support for Dynamic and DLR Enhanced support for COM interop |
| 5.0 | 2012 | Visual Studio 2012 | Async / Await Feature Support for caller information |
| 6.0 | 2015 | Visual Studio 2015 |  |

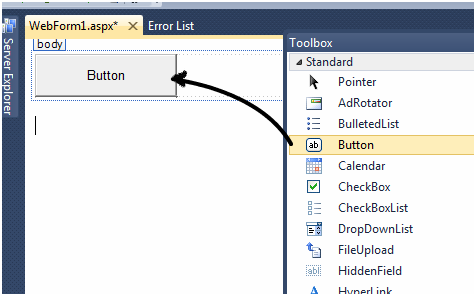
**.Net Architecture**

If you watch the recent Microsoft agenda you will clearly see they are focusing on MVC , MVC and MVC. So the question is why is Microsoft so keen to dump a successful thing like ASP.NET Webform and persuade the Microsoft web development community to use ASP.NET MVC.

That’s what this article will focus on.

ASP.NET Webform behind code –boon and curse

If you closely watch ASP.NET webform technology it’s a RAD / VISUAL approach for web development. In other words developers drag and drop user controls on the designer and the VS tool codes in the behind code.



So in other words when you drag and drop a button control on the designer a button object is created and developers can write code in the click event of the button object.

Hide   Copy Code

public partial class WebForm1 : System.Web.UI.Page

{

protectedvoidPage\_Load(object sender, EventArgs e)

{

// Developers write code here

}

protectedvoid Button1\_Click(object sender, EventArgs e)

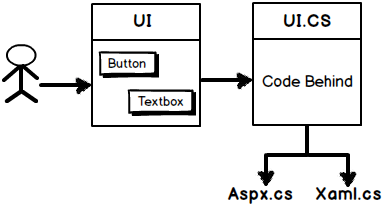
{

// Developers write code here

}

}

So when developers drag and drop these UI elements, double click to get the events on the front end, write logic in those events etc. At the back end smartly and quietly Microsoft code’s logic in the ASPX.CS partial class file.



Now this partial behind code file was the key to success for delivering ASP.NET Webformprojects faster as developers where encapsulated from lot of technical details like events, delegates, HTTP protocol POST,GET, session management etc. You would probably like to read this article [why Microsoft has partial classes ?](http://www.codeproject.com/Articles/819841/Why-Microsoft-has-partial-classes-and-Java-does-no)and Microsoft UI success story.

But due to the way the behind code was positioned and invoked it has 5 serious problems. So let’s discuss those 5 problems and how MVC helps to address the same.

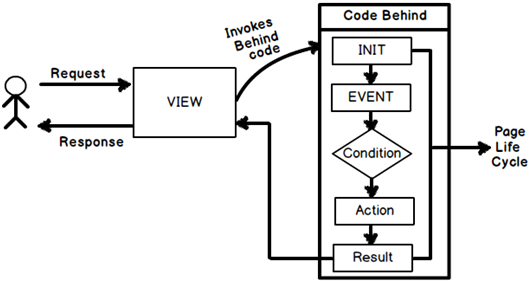
**Problem 1 :- View based solution for Action based requirement**

Websites at the end of the day are used by end users. End users come with a specific purpose to a website and they communicate their purpose by actions. For instance if somebody comes to shop on a shopping portal he will communicate his purposeusing actions like:-

* Buy product.
* Print invoice

Now these actions are communicated by button click , right click or through a browser URL etc. Due to this action based structure HTTP protocol was chosen for Web because it had actions like POST,GET,PUT,DELETEetc which can communicate the purpose of end users more clearly. That also makes REST a popular way of addressing end user request. So logically if we can map these actions to methods / functions of our program that would make more sense and also keep the architecture simple.

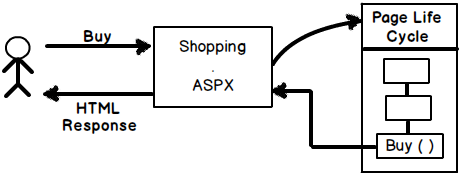
But Microsoft had no way out, they wanted to support RAD concept or we can term it as visual programming concept, so they ended up with a view based solution for an action based structure.



So flow of request went something WEIRD like this for web forms( for visual’s see the above diagram):-

* End user sends a request with action like HTTP POST / GET etc.
* IIS webserver maps this request to a view.
* View invokes the page life cycle, walks through events and then **INVOKES THE APPROPRIATE ACTION**.
* Finally action put’s the **RESULT IN HTML** format and sends it to the end user browser.

Microsoft ended with a view based architecture for an action based requirement. So the architecture itself was not fitting logically to the end user’s action based approach. In other words if the end user sends a “Buy” action it first comes to a view like “Shopping.aspx” who in turn kicks of “Shopping.aspx.cs” which executes a complicated page life cycle which in turn executes the action which will fulfill the request of the end user.



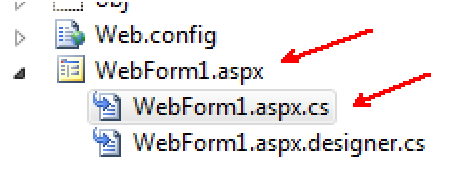
This is like hitting the bush. The end requests are getting mapped to the actual action after a complicated page life cycle is completed. So how about we make it an action oriented architecture rather than view oriented. Or I can rephrase it **“HOW CAN WE MAKE ACTION FIRST STRUCTURE RATHER THAN A VIEW FIRST STRUCTURE ?”**.

So how about hitting the action first and then the action picks up view. This would make the flow more logical and clear. That’s what exactly MVC architecture does. The first hit comes to an action which belongs in to a controller and then controller invokes the view with appropriate model.

http://www.codeproject.com/KB/aspnet/821275/6.png

**Problem 2:- Side effects of bad architecture: - Tight coupling**

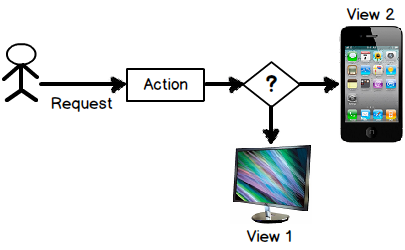
Once you start with a wrong architecture you end up adjusting things and then you end up with serious side effects. In this case the same was happening. The behind code which looks physically different in different files was never actually decoupled i.e. ASPX.CS cannot be separated from ASPX.



In simple words I cannot attach “Customer.aspx.cs” with “CustomerDetailed.aspx” easily. The behind code is tightly coupled with view. It is not reusable.

If you ever analyze the quantity of behind code w.r.t to other layers of the project it’s huge in size with complex events. This makes the code unreadable and difficult to maintain from long term perspective.

So if we can change the view first based architecture to action first based architecture then we can reuse the same action code with different views. For instance if an end user sends an action “Display” it can invoke “DisplayDesktop.aspx” or it can display “DisplayMobile.aspx” depending on the type of device.



So in the MVC action depending on situation we can invoke “MobileView” or “NormalView” , below is the sample code for the same. Now imagining achieving this in behind code of the Webform , difficult very difficult right.

Hide   Copy Code

publicActionResult Index(string DeviceType)

{

if (viewType == "Mobile")

{

return View("MobileView");

}

else

{

return View("NormalView");

}

}

**Problem 3:- HTML is not the only response type**

Because of the tight coupling between view and code behind even the response type is fixed in webform , its by default HTML. If you wish to change it you need to play around with Content-type and “Response.End” methods etc which is quiet tedious.

If we create “Action” first structure then the action has all the luxury in the world to decide what kind of response type should go out. This makes our system more flexible in terms of same action with different outputs.

Below is a simple MVC action code which send’s JSON or HTML result depending on the value passed to the action. This kind of flexibility is difficult to achieve with webformview’s because they are meant to emit only HTML.

Hide   Copy Code

publicActionResult Index(string viewType)

{

if (viewType == "JSON")

{

returnJson(new Customer(), JsonRequestBehavior.AllowGet);

}

else

{

return View("DisplayCustomer", new Customer());

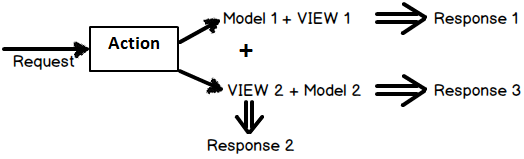
}

}

**Problem 4:- Flexible Combination of view and data**

When we send response to a user it’s a combination of view (display) and data (model).Webform is a view first architecture so the view decides which model to connect making the view NOT SO flexible and also involving view in complex decision making. That clearly violates **SRP of the SOLID principle** read here more about [SOLID Principles](http://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp).

But if we make action first architecture then the first hits comes to the action and he picks the Model and the view to send different responses.



In MVC action you can code something as shown below. You can pick up the same model and attach it with different view. For example in the below code we have taken “customerdata” model and attached with “DetailCustomer” view and the same model in other situation is attached with “Customer” view.

Hide   Copy Code

publicActionResult Index(string ViewName,Customercustomerdata)

{

if (ViewName == "Detailed")

{

return View("DetailCustomer",customerdata);

}

else

{

return View("Customer",customerdata);

}

}

This kind of flexibility to achieve through Webform is very difficult because the invocation comes on the view itself and then you need write all decision making logic in the page life cycle and redirect to some other view not making the implementation so clean.

**Problem 5:- Making behind code a normal class for unit testing**

The behind code in webform is a very typical heavy and bulky weight partial class which cannot be instantiated in simple C# code straightforward. Remember the Webform screen inherits from the “Page” class. This page class cannot be created directly as it has lot of dependencies.

Hide   Copy Code

public partial class WebForm1 : System.Web.UI.Page

{

protectedvoidPage\_Load(object sender, EventArgs e)

{

}

publicvoid Button1\_Click(object sender, EventArgs e)

{

Session["SomeSession"] = "Is this set";

}

}

Now the next thing which would come to your mind why would you want this page class to be instantiated. One of the place where I would like this page class to be instantiated is for unit testing. I would like to invoke the actions of the button click methods and test if the session variables are set properly,view states are properly etc.

But if you ever try to do that like as shown in the below code you will end with weird code as shown below. Watch out those ugly event args passed to the button click methods.

Hide   Copy Code

[TestMethod]

publicvoid TestMethod1()

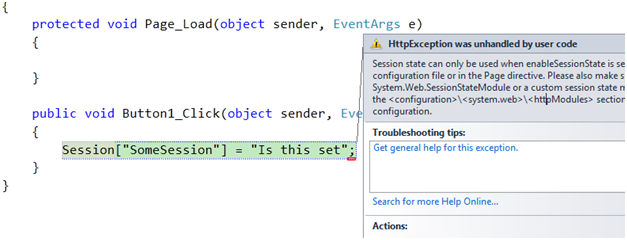
{

WebApplication22.WebForm1 obj = newWebApplication22.WebForm1();

obj.Button1\_Click(this, newEventArgs());

}

And when you invoke it asks for more things which makes the UI Unit testing impossible.



In case of MVC this becomes a normal class. A class which can instantiated in simple unit test project and you can test various aspects like session ,viewbag , tempdata in an easy way.

Hide   Copy Code

publicclassHomeController : Controller // &szlig; this class is simple

{

publicActionResult Index()

{

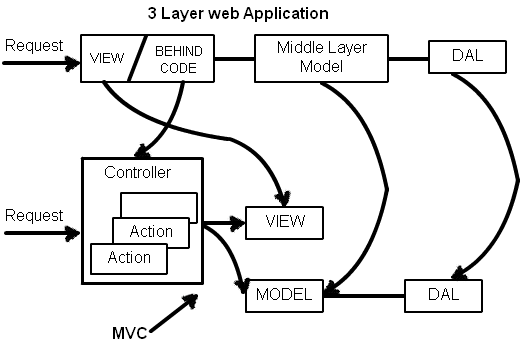
Session["SomeSession"] = "Is this set";

return View("SomeView");

}

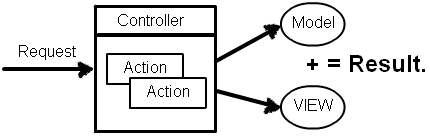
}

So the solution is MVC?



So to change from a view based architecture to an action based MVC architecture we need to do the following structural changes ( The above image gives a visual view of the same):-

* Move the behind code in to a controller class with all the events converted in to methods which can be termed as actions.
* The middle layer becomes the model which provides data and business logic.
* The view just does display,positioning and layouting.
* The DAL and other layers do not change a lot as they have no connection with the behind code issue directly.



So with MVC architecture we have the below three step flow:-

* End user sends his request application. Application routes the request to the Controller. Controller is a logical entity which groups actions together.
* Controller maps that request to a particular action.
* Now action has two tasks to do first it needs to access appropriate data depending on the action and second that data has to be connected to the proper view. Action creates the model object and connects the model to the view to send the final response.

# Memory management

### Managed code vs unmanaged code

* + **Managed Code**  
      
    Managed code is code that is written to target the services of the managed runtime execution environment (like Common Language Runtime in .NET Framework). The managed code is always executed by a managed runtime execution environment rather than the operating system directly. Managed refers to a method of exchanging information between the program and the runtime environment. Because the execution of code is governed by the runtime environment, the environment can guarantee what the code is going to do and provide the necessary security checks before executing any piece of code. Because of the same reason the managed code also gets different services from the runtime environment like [Garbage Collection](http://www.codeguru.com/forum/showthread.php?t=368537), type checking, exception handling, bounds checking, etc. This way managed code does not have to worry about memory allocations, type safety, etc. Applications written in Java, C#, VB.NET, etc target a runtime environment which manages the execution and the code written using these types of languages is known as Managed Code. Managed code is always compiled into an Intermediate Language (MSIL in case of .NET Framework). The compiler used by .NET framework to compile managed code compiles it into Intermediate Language and generates the necessary [metadata](http://www.codeguru.com/forum/showthread.php?t=369067), symbolic information that describes all of the entry points and the constructs exposed in the Intermediate Language (e.g., methods, properties) and their characteristics. The [Common Language Infrastructure](http://www.codeguru.com/forum/showthread.php?t=368666) (CLI) Standard describes how the information is to be encoded, and programming languages that target the runtime emit the correct encoding.   
      
    In .NET Framework Managed Code runs within the .Net Framework’s CLR and benefits from the services provided by the CLR. When we compile the managed code, the code gets compiled to an intermediate language (MSIL) and an executable is created. When a user runs the executable the Just In Time Compiler of CLR compiles the intermediate language into native code specific to the underlying architecture. Since this translation happens by the managed execution environment (CLR), the managed execution environment can make guarantees about what the code is going to do, because it can actually reason about it. It can insert traps and sort of protection around, if it's running in a sandboxed environment, it can insert all the appropriate garbage collection hooks, exception handling, type safety, array bounce, index checking and so forth.   
      
    Managed code also provides platform independence. As the managed code is first compiled to intermediate language, the CLR’s JIT Compiler takes care of compiling this intermediate language into the architecture specific instructions.
  + **Unmanaged Code**  
      
    Code that is directly executed by the Operating System is known as un-managed code. Typically applications written in VB 6.0, C++, C, etc are all examples of unmanaged code. Unmanaged code typically targets the processor architecture and is always dependent on the computer architecture. Unmanaged code is always compiled to target a specific architecture and will only run on the intended platform. This means that if you want to run the same code on different architecture then you will have to recompile the code using that particular architecture. Unmanaged code is always compiled to the native code which is architecture specific. When we compile unmanaged code it gets compiled into a binary X86 image. And this image always depends on the platform on which the code was compiled and cannot be executed on the other platforms that are different that the one on which the code was compiled. Unmanaged code does not get any services from the managed execution environment.  
      
    In unmanaged code the memory allocation, type safety, security, etc needs to be taken care of by the developer. This makes unmanaged code prone to memory leaks like buffer overruns and pointer overrides and so forth.  
      
    Unmanaged executable files are basically a binary image, x86 code, loaded into memory. The program counter gets put there and that’s the last the Operating System knows. There are protections in place around memory management and port I/O and so forth, but the system doesn’t actually know what the application is doing.

### [**Conditions for a garbage collection**](javascript:void(0))

Garbage collection occurs when one of the following conditions is true:

* The system has low physical memory.
* The memory that is used by allocated objects on the managed heap surpasses an acceptable threshold. This threshold is continuously adjusted as the process runs.
* The [GC.Collect](https://msdn.microsoft.com/en-us/library/system.gc.collect(v=vs.110).aspx) method is called. In almost all cases, you do not have to call this method, because the garbage collector runs continuously. This method is primarily used for unique situations and testing.

## [Generations](javascript:void(0))

The heap is organized into generations so it can handle long-lived and short-lived objects. Garbage collection primarily occurs with the reclamation of short-lived objects that typically occupy only a small part of the heap. There are three generations of objects on the heap:

* Generation 0. This is the youngest generation and contains short-lived objects. An example of a short-lived object is a temporary variable. Garbage collection occurs most frequently in this generation.

Newly allocated objects form a new generation of objects and are implicitly generation 0 collections, unless they are large objects, in which case they go on the large object heap in a generation 2 collection.

Most objects are reclaimed for garbage collection in generation 0 and do not survive to the next generation.

* Generation 1. This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.
* Generation 2. This generation contains long-lived objects. An example of a long-lived object is an object in a server application that contains static data that is live for the duration of the process.

Garbage collections occur on specific generations as conditions warrant. Collecting a generation means collecting objects in that generation and all its younger generations. A generation 2 garbage collection is also known as a full garbage collection, because it reclaims all objects in all generations (that is, all objects in the managed heap).

### [Survival and promotions](javascript:void(0))

Objects that are not reclaimed in a garbage collection are known as survivors, and are promoted to the next generation. Objects that survive a generation 0 garbage collection are promoted to generation 1; objects that survive a generation 1 garbage collection are promoted to generation 2; and objects that survive a generation 2 garbage collection remain in generation 2.

When the garbage collector detects that the survival rate is high in a generation, it increases the threshold of allocations for that generation, so the next collection gets a substantial size of reclaimed memory. The CLR continually balances two priorities: not letting an application's working set get too big and not letting the garbage collection take too much time.

## [What happens during a garbage collection](javascript:void(0))

A garbage collection has the following phases:

* A marking phase that finds and creates a list of all live objects.
* A relocating phase that updates the references to the objects that will be compacted.
* A compacting phase that reclaims the space occupied by the dead objects and compacts the surviving objects. The compacting phase moves objects that have survived a garbage collection toward the older end of the segment.

Because generation 2 collections can occupy multiple segments, objects that are promoted into generation 2 can be moved into an older segment. Both generation 1 and generation 2 survivors can be moved to a different segment, because they are promoted to generation 2.

# **Cleaning Up Unmanaged Resources**

* [.NET Framework 4](https://msdn.microsoft.com/en-us/library/498928w2(v=vs.100).aspx)
* [.NET Framework 3.0](https://msdn.microsoft.com/en-us/library/498928w2(v=vs.85).aspx)
* [.NET Framework 1.1](https://msdn.microsoft.com/en-us/library/498928w2(v=vs.71).aspx)
* [.NET Framework 3.5](https://msdn.microsoft.com/en-us/library/498928w2(v=vs.90).aspx)
* [.NET Framework 2.0](https://msdn.microsoft.com/en-us/library/498928w2(v=vs.80).aspx)

For the majority of the objects that your app creates, you can rely on the .NET Framework's garbage collector to handle memory management. However, when you create objects that include unmanaged resources, you must explicitly release those resources when you finish using them in your app. The most common types of unmanaged resource are objects that wrap operating system resources, such as files, windows, network connections, or database connections. Although the garbage collector is able to track the lifetime of an object that encapsulates an unmanaged resource, it doesn't know how to release and clean up the unmanaged resource.

* Implement the [dispose pattern](https://msdn.microsoft.com/en-us/library/b1yfkh5e(v=vs.110).aspx). This requires that you provide an [IDisposable.Dispose](https://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) implementation to enable the deterministic release of unmanaged resources. A consumer of your type calls [Dispose](https://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) when the object (and the resources it uses) is no longer needed. The [Dispose](https://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method immediately releases the unmanaged resources.

**Disadvantages of Finalize method**

If a type does override the Finalize method, the garbage collector adds an entry for each instance of the type to an internal structure called the finalization queue. The finalization queue contains entries for all the objects in the managed heap whose finalization code must run before the garbage collector can reclaim their memory. The garbage collector then calls the Finalize method automatically under the following conditions:

* After the garbage collector has discovered that an object is inaccessible, unless the object has been exempted from finalization by a call to the [GC.SuppressFinalize](https://msdn.microsoft.com/en-us/library/system.gc.suppressfinalize(v=vs.110).aspx) method.
* During shutdown of an application domain, unless the object is exempt from finalization. During shutdown, even objects that are still accessible are finalized.

Finalize operations have the following limitations:

* The exact time when the finalizer executes is undefined. To ensure deterministic release of resources for instances of your class, implement a Close method or provide a[IDisposable.Dispose](https://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) implementation.
* The finalizers of two objects are not guaranteed to run in any specific order, even if one object refers to the other. That is, if Object A has a reference to Object B and both have finalizers, Object B might have already been finalized when the finalizer of Object A starts.
* The thread on which the finalizer runs is unspecified.