1. **.NET Fundamentals**
   1. **History**

The .NET Framework is a software development platform of Microsoft .NET. Like any platform, it provides a runtime, defines functionality in some libraries, and supports a set of programming languages. The .NET Framework provides the necessary compile-time and run-time foundation to build and run .NET-based applications.

* Starts in late 1990s originally under the name of Next Generation Windows Services (NGWS)
* 2001- the first beta versions of .NET 1.0 were released.
* 13 February 2002 - the first version of .NET Framework was released
* Latest 4.7.2 (2018)
* Developing (4.7.3 and 4.8)
* .NET Core - open source, 1.0 released in 2016-JUN-27, latest 2.1 released in 2018
* Visual Studio 2017 is the latest version and 2019 is on its way
  1. **Overview**
* platform neutral framework.
* layer between the operating system and the programming language.
* supports many programming languages, including VB.NET, C# etc.
* .NET provides a common set of class libraries, which can be accessed from any .NET based programming language. There will not be separate set of classes and libraries for each language. If you know any one .NET language, you can write code in any .NET language
* In newer versions of Windows, .NET is freely distributed as part of operating

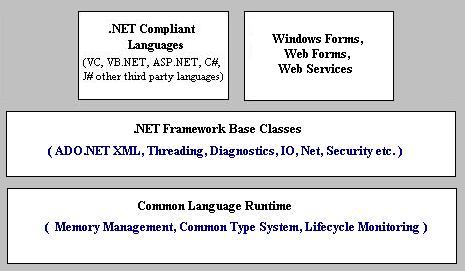


Fig: Major Components of .NET Framework

* 1. **Advantages**
* It allows the use of multiple languages
* It has horizontal scalability
* Security
* .NET creates a unified environment that allows developers to create programs in C++, Java or Visual Basic
* Interfaces easily with Windows or Microsoft
* All tools and IDEs have been pre-tested and are easily available in the Microsoft Developer Network.
* UI best practices are more consistent
* Language integration is seamless, as you can call methods from C# to VB.NET

1. **Windows Development using .NET**
   1. **Overview – Structured programming and object-oriented programming**

Structured Programming Language

* Follow top-down approach to program design.
* Data and Functions don’t tide with each other.
* Large programs are divided into smaller self-contained program segment known as functions.
* Data moves openly around the system from function to function.
* Functions are dependent so reusability is not possible

Object Oriented Programming Language

* Follow bottom-up approach in program design.
* Functions and data are tied together.
* Programs are divided into entity called Objects.
* Data is hidden and can’t be accessed by the external world
* Functions are not dependent so reusability is possible
  1. **C#**

In the .NET framework, Microsoft included a new language called C# (pronounced C Sharp). C# is designed to be a simple, modern, general-purpose, object-oriented, and type safe programming language derived from C and C++ programming language.

* 1. **Windows Forms**

Windows Forms (WinForms) is a graphical (GUI) class library included as a part of Microsoft .NET Framework, providing a platform to write rich client applications for desktop, laptop, and tablet PCs. Windows Forms offers an extensive client library providing interface to access native Windows graphical interface elements and graphics from managed code. A Windows Forms application is an event-driven application. Unlike a batch program, it spends most of its time simply waiting for the user to do something, such as fill in a text box or click a button. A form is ultimately a blank slate that a developer can enhance with controls to create a user interface and with code to manipulate data.

* 1. **User Controls**

User Controls are objects that are contained within form objects. Each type of control has its own set of properties, methods, and events that make it suitable for a particular purpose. User Controls can be manipulated in the designer or write code to add controls dynamically at run time.

All visual elements in the Windows Forms class library derive from the Control class. This provides a minimal functionality of a user interface element such as location, size, colour, font, text, as well as common events like click and drag/drop. The Control class also has docking support to let a control rearrange its position under its parent. The Microsoft Active Accessibility support in the Control class also helps impaired users to use Windows Forms better.

Native Windows controls are button, textbox, checkbox and listview.

1. **Web Development using .NET**
   1. **Overview – HTML, JavaScript, ASP**

ASP: - Stands for Active Server Pages, is Microsoft's first server-side script engine for dynamically generated web pages. It is a server-side technology. When a browser requests an ASP or ASP.NET file, the ASP engine reads the file, executes any code in the file, and returns the result to the browser.

* 1. **ASP.NET Web Applications**
* Classic ASP
* ASP .NET
* ASP.NET MVC
* ASP.NET Web API
* ASP.NET Web Forms
* ASP.NET Core
  1. **Web Forms**

ASP.NET Web Forms pages allow you to create dynamic content for your Web application. With a static HTML page (.htm or .html file), the server fulfils a Web request by reading the file and sending it as-is to the browser. In contrast, when someone requests an ASP.NET Web Forms page (.aspx file), the page runs as a program on the Web server. While the page is running, it can perform any task that your Web site requires, including calculating values, reading or writing database information, or calling other programs. As its output, the page dynamically produces mark-up (elements in HTML or another mark-up language) and sends this dynamic output to the browser.

ASP.NET Web Forms pages run as code on the server. Therefore, for the page to be processed, the page is configured to submit to the server when users click buttons (or optionally, when users select check boxes or interact with other controls in the page). Each time, the page is submitted back to itself so it can run its server code again and then render a new version of itself back to the user.

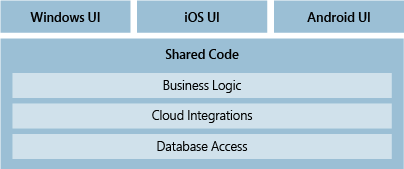
The processing cycle for an ASP.NET Web Forms page is this:

* The user requests the .aspx page. (The page is requested using an HTTP GET method.) The page runs for the first time, performing preliminary processing if you have programmed it to do so.
* The page dynamically renders markup to the browser, which the user sees as a Web page similar to any other page.
* The user types information or selects from available choices and then clicks a button. (If users click a link instead of a button, the page might simply navigate to another page, and no further processing takes place on the first page.)
* The page is posted to the Web server. (The browser performs an HTTP POST method, which in ASP.NET is referred to as a postback.) Specifically, the page is posted back to itself. For example, if the user is working with the page Default.aspx, clicking a button on the page posts the page back to the server with a target of Default.aspx.
* On the Web server, the page runs again. The information that the user typed or selected is available to the page.
* The page performs the processing that you have programmed it to do.
* The page renders itself back to the browser.

This cycle continues as long as the user is working in the page. Each time the user clicks a button, the information in the page is posted to the Web server and the page runs again. Each cycle is referred to as a round trip. Because page processing occurs on the Web server, each action that the page can do requires a round trip to the server.

* 1. **ASP.NET Mobile applications**

Mobile application development in ASP.NET is similar to traditional ASP.NET web application development. And it is very easy for ASP.NET developer to develop mobile application. All mobile web pages are inherit from **MobilePage** class which exists in **System.Web.UI.MobileControls** namespace.ASP.NET exposes a **System.Web.Mobile** namespace is for specifically to Web development. Visual studio provides the SDK and Emulator for android as well as IOS devices.



* Windows Phone Application
* Hybrid Application development with Xamarin
* Universal Application Development
* Mobile friendly and Mobile targeted web applications

1. **Web Services using .NET**
   1. **Overview – Web Services and their applications**

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards. Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

* 1. **XML**
     + XML stands for eXtensible Markup Language
     + XML is a markup language much like HTML
     + XML was designed to store and transport data
     + XML was designed to be self-descriptive
     + XML is a W3C Recommendation

XML does not DO anything. XML is just information wrapped in tags. Someone must write a piece of software to send, receive, store, or display it. The XML language has no predefined tags. XML is Extensible i.e. Most XML applications will work as expected even if new data is added (or removed). The basic building block of an XML document is an element, defined by tags. An element has a beginning and an ending tag. All elements in an XML document are contained in an outermost element known as the root element. XML can also support nested elements, or elements within elements. This ability allows XML to support hierarchical structures. Element names describe the content of the element, and the structure describes the relationship between the elements.

* **Tag**: A tag is a markup construct that begins with < and ends with >. An example is <section>.
* **Element**: An element is a logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start-tag and end-tag, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example is <greeting>Hello, world! </greeting>.
* **Attribute**: An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag. An example is <img src="madonna.jpg" alt="Madonna" />.
* **XML declaration**: XML documents may begin with an *XML declaration* that describes some information about themselves. An example is <?xml version="1.0" encoding="UTF-8"?>.

XML Schema and Validation

* **Document Type Definition:** A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference. A DTD is associated with an XML document by means of a document type declaration (DOCTYPE). The DOCTYPE appears in the syntactic fragment *doctypedecl* near the start of an XML document. The declaration establishes that the document is an instance of the type defined by the referenced DTD.

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE note SYSTEM "Note.dtd">  
<note>  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don't forget me this weekend!</body>  
</note>

Note.dtd

<!DOCTYPE note  
[  
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>  
]>

* **XML Schema:** An XML Schema describes the structure of an XML document, just like a DTD. An XML document with correct syntax is called "Well Formed". An XML document validated against an XML Schema is both "Well Formed" and "Valid". It supports data types, uses XML syntax and restrictions on value and data types.

<note xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance> xsi:schemaLocation="note.xsd">

<xs:element name="note">

<xs:complexType>  
  <xs:sequence>  
    <xs:element name="to" type="xs:string"/>  
    <xs:element name="from" type="xs:string"/>  
    <xs:element name="heading" type="xs:string"/>  
    <xs:element name="body" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>  
</xs:element>

* 1. **XML Web Services**

XML Web services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web services are becoming the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented. One of the primary advantages of the XML Web services architecture is that it allows programs written in different languages on different platforms to communicate with each other in a standards-based way.

* 1. **SOAP, WSDL and UDDI**
  + **SOAP (Simple Object Access Protocol)**

Soap is the communications protocol for XML Web services. SOAP is a specification that defines the XML format for messages. a well-formed XML fragment enclosed in a couple of SOAP elements is a SOAP message. Its purpose is to induce extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using XML. Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/" soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>...</soap:Header>

<soap:Body>

...

<soap:Fault>...</soap:Fault>

</soap:Body>

</soap:Envelope>

* + **WSDL (Web Services Description Language)**

WSDL is used to describe web services. Since WSDL is XML, it is readable and editable but, in most cases, it is generated and consumed by software. WSDL is written in XML. The notation that a WSDL file uses to describe message formats is based on the XML Schema standard which means it is both programming-language neutral and standards-based which makes it suitable for describing XML Web services interfaces that are accessible from a wide variety of platforms and programming languages. In addition to describing message contents, WSDL defines where the service is available and what communications protocol is used to talk to the service. This means that the WSDL file defines everything required to write a program to work with an XML Web service. An WSDL document specifies the location of the service, and the methods of the service, using these major elements:

|  |  |
| --- | --- |
| 1. <types> | Defines the (XML Schema) data types used by the web service |
| 2. <message> | Defines the data elements for each operation |
| 3. <portType> | Describes the operations that can be performed and the messages involved. |
| 4. <binding> | Defines the protocol and data format for each port type |

<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>  
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>  
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>

<binding type="glossaryTerms" name="b1">  
   <soap:binding style="document"  
   transport="http://schemas.xmlsoap.org/soap/http" />  
   <operation>  
     <soap:operation soapAction="http://example.com/getTerm"/>  
     <input><soap:body use="literal"/></input>  
     <output><soap:body use="literal"/></output>  
  </operation>  
</binding>

* + **UDDI (Universal Description, Discovery and Integration)**

UDDI is an XML-based standard for describing, publishing, and finding web services. UDDI is a specification for a distributed registry of web services which is a platform-independent, open framework. It can communicate via SOAP, CORBA, Java RMI Protocol and uses Web Service Definition Language (WSDL) to describe interfaces to web services. UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.

<bindingTemplate serviceKey = "uuid: D6F1B765-BDB3-4837-828D-8284301E5A2A"

bindingKey = "uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">

<description>Hello World SOAP Binding</description>

<accessPoint URLType = "http">http://localhost:8080</accessPoint>

<tModelInstanceDetails>

<tModelInstanceInfo tModelKey = "uuid: EB1B645F-CF2F-491f-811A-4868705F5904">

<instanceDetails>

<overviewDoc>

<description>

references the description of the WSDL service definition

</description>

<overviewURL>

http://localhost/helloworld.wsdl

</overviewURL>

</overviewDoc>

</instanceDetails>

</tModelInstanceInfo>

</tModelInstanceDetails>

</bindingTemplate>

1. **Database Development using .NET**
   1. **Overview – Database concepts and SQL**
   2. **ADO.NET**
   3. **XML and Data**
2. **.NET Common Language Runtime**
   1. **Overview**

The .NET Framework provides a run-time environment called the common language runtime, which runs the code and provides services that make the development process easier.

Compilers and tools expose the common language runtime's functionality and enable you to write code that benefits from this managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code; it benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

The runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. Objects whose lifetimes are managed in this way are called managed data. Garbage collection eliminates memory leaks as well as some other common programming errors. If your code is managed, you can use managed data, unmanaged data, or both managed and unmanaged data in your .NET Framework application. Because language compilers supply their own types, such as primitive types, you might not always know (or need to know) whether your data is being managed.

The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be tightly integrated. For example, you can define a class and then use a different language to derive a class from your original class or call a method on the original class. You can also pass an instance of a class to a method of a class written in a different language. This cross-language integration is possible because language compilers and tools that target the runtime use a common type system defined by the runtime, and they follow the runtime's rules for defining new types, as well as for creating, using, persisting, and binding to types.

The CLR provides a number of services that include:

* Loading and execution of codes
* Memory isolation for application
* Verification of type safety
* Compilation of IL into native executable code
* Providing metadata
* Automatic garbage collection
* Enforcement of Security
* Interoperability with other systems
* Managing exceptions and errors
* Provide support for debugging and profiling
  1. **Assemblies and JIT**

An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.

In the .NET Framework, all the Microsoft .NET languages use a Common Language Runtime, which solves the problem of installing separate runtimes for each of the programming languages. When the Microsoft .NET Common Language Runtime is installed on a computer then it can run any language that is Microsoft .NET compatible. Before the Microsoft Intermediate Language (MSIL) can be executed, it must be converted by a .NET Framework Just-In-Time (JIT) compiler to native code, which is CPU-specific code that runs on the same computer architecture as the JIT compiler. Rather than using time and memory to convert all the MSIL in portable executable (PE) file to native code, it converts the MSIL as it is needed during execution and stored in resulting native code so it is accessible for subsequent calls.

There are three types of JIT Compilers

* Pre-JIT Compiler (Compiles entire code into native code completely)
* Econo JIT Compiler (Compiles code part by part freeing when required)
* Normal JIT Compiler (Compiles only that part of code when called and places in cache
  1. **Garbage Collection**

The Garbage collection is the technique in the .Net framework to free the unused managed code objects in the memory and free the space to the process. In the common language runtime (CLR), the garbage collector serves as an automatic memory manager. Each process has its own, separate virtual address space. Since, all processes on the same computer share the same physical memory, and share the page file if there is one, the garbage collector allocates and frees virtual memory for you on the managed heap. It enables users to develop application without having to free memory and allocates objects on the managed heap efficiently. Garbage Collection Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Managed objects automatically get clean content to start with, so their constructors do not have to initialize every data field.

* 1. **Security**

The common language runtime and the .NET Framework provide many useful classes and services that enable developers to easily write secure code and enable system administrators to customize the permissions granted to code so that it can access protected resources. In addition, the runtime and the .NET Framework provide useful classes and services that facilitate the use of cryptography and role-based security. The Microsoft .NET Framework offers security transparency, code access security and role-based security to help address security concerns about mobile code and to provide support that enables components to determine what users are authorized to do. These security mechanisms use a simple, consistent model so that developers familiar with code access security can easily use role-based security, and vice versa. Both code access security and role-based security are implemented using a common infrastructure supplied by the common language runtime. The common language runtime allows code to perform only those operations that the code has permission to perform. The runtime uses objects called permissions to enforce restrictions on managed code. The runtime provides built-in permission classes in several namespaces and also supports designing and implementing custom permission classes. Type-safe code accesses only the memory locations it is authorized to access. During just-in-time (JIT) compilation, an optional verification process examines the metadata and Microsoft intermediate language (MSIL) of a method to be JIT-compiled into native machine code to verify that they are type safe. This process is skipped if the code has permission to bypass verification.

* 1. **Events and Delegates**

An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The Event model in C# finds its roots in the event programming model that is popular in asynchronous programming. The basic foundation behind this programming model is the idea of "publisher and subscribers." In this model, you have publishers who will do some logic and publish an "event." Publishers will then send out their event only to subscribers who have subscribed to receive the specific event. In C#, any object can publish a set of events to which other applications can subscribe. When the publishing class raises an event, all the subscribed applications are notified.

The following important conventions are used with events:

* Event Handlers in the .NET Framework return void and take two parameters.
* The first parameter is the source of the event; that is the publishing object.
* The second parameter is an object derived from EventArgs.
* Events are properties of the class publishing the event.
* The keyword event controls how the event property is accessed by the subscribing classes.

A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback. While delegates have other uses, the discussion here focuses on the event handling functionality of delegates. A delegate declaration is sufficient to define a delegate class. The declaration supplies the signature of the delegate, and the common language runtime provides the implementation. In C#, delegates are multicast, which means that they can point to more than one function at a time (that is, they're based off the **System.MulticastDelegate** type). A multicast delegate maintains a list of functions that will all be called when the delegate is invoked.

* 1. **Reflection**

Reflection objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in the System.Reflection namespace. The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application. Reflection provides objects that encapsulate assemblies, modules and types. A program reflects on itself by extracting metadata from its assembly and using that metadata either to inform the user or to modify its own behavior. We can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.

* 1. **Remoting**

Remoting is a framework built into Common Language Runtime (CLR) in order to provide developers classes to build distributed applications and wide range of network services. .NET Remoting is a mechanism for communicating between objects which are not in the same process. It is a generic system for different applications to communicate with one another. .NET objects are exposed to remote processes, thus allowing inter process communication. The applications can be located on the same computer, different computers on the same network, or on computers across separate networks. Microsoft .NET Remoting provides a framework that allows objects to interact with each other across application domains. Remoting was designed in such a way that it hides the most difficult aspects like managing connections, marshaling data, and reading and writing XML and SOAP. The framework provides a number of services, including object activation and object lifetime support, as well as communication channels which are responsible for transporting messages to and from remote applications.

* 1. **CLS and Interop**

The language interoperability, and .NET Class Framework, are not possible without all the language sharing the same data types. What this means is that an int should mean the same in VB, VC++, C# and all other .NET compliant languages. Same idea follows for all the other data types. This is achieved through introduction of Common Type System (CTS). CTS, defines every data type as a Class. Every .NET compliant language must stick to this definition. Since CTS defines every data type as a class; this means that only Object-oriented (or Object-Based) languages can achieve .NET compliance.

To fully interact with other objects regardless of the language they were implemented in, objects must expose to callers only those features that are common to all the languages they must interoperate with. For this reason, the Common Language Specification (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the Common Type System; that is, all the rules that apply to the common type system apply to the CLS, except where stricter rules are defined in the CLS. The CLS helps enhance and ensure language interoperability by defining a set of features that developers can rely on to be available in a wide variety of languages. The CLS also establishes requirements for CLS compliance; these help user to determine whether the managed code conforms to the CLS and to what extent a given tool supports the development of managed code that uses CLS features.

COM Interop is a technology included in the .NET Framework Common Language Runtime (CLR) that enables Component Object Model (COM) objects to interact with .NET objects, and vice versa. COM Interop aims to provide access to the existing COM components without requiring that the original component be modified. It tries to make the .NET types equivalent to the COM types. In addition, COM Interop allows COM developers to access managed objects as easily as they access other COM objects.

1. **.NET Framework Class Library**
   1. **Overview**

The .NET Framework class library is a library of classes, interfaces, and value types that provide access to system functionality. It is the foundation on which .NET Framework applications, components, and controls are built and provides the core functionality of .Net Framework architecture. NET implementations include classes, interfaces, delegates, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, most .NET types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS). The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies. The System Namespace is the root for types in the .NET Framework. The .Net FCL classes are managed classes that provide access to System Services. The .Net FCL classes are object oriented and easy to use in program developments. Moreover, third-party components can integrate with the classes in the .NET Framework.

* 1. **GDI+**

Stands for Graphics Device Interface which is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers. GDI+ is next evolution of GDI. GDI+ consists of the set of .NET base classes that are available for the purpose of carrying out custom drawing on the screen. These classes are able to arrange for the appropriate instructions to be sent to the graphics device drivers to ensure the correct output is placed on the monitor screen (or printed to a hard copy). Just as for the rest of the .NET base classes, the GDI+ classes are based on a very intuitive and easy to use object model. All GDI+ classes are reside in the **System.Drawing, System.Text, System.Printing, System.Internal, System.Imaging, System.Drawing2D** and **System.Design** namespaces. GDI's most significant advantages over more direct methods of accessing the hardware are perhaps its scaling capabilities and its abstract representation of target devices. Using GDI, it is very easy to draw on multiple devices, such as a screen and a printer, and expect proper reproduction in each case. This capability is at the center of most What You See Is What You Get applications for Microsoft Windows

* 1. **Exceptions**

An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally, and throw.

* **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
* **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
* **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
* **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class are the **System.ApplicationException** and **System.SystemException** classes.

e.g.: IOException, IndexOutOfRangeException, NullReferenceException, InvalidCastException, InvalidCastException

* 1. **Input/Output**

A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream. The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the input stream and the output stream. The input stream is used for reading data from file (read operation) and the output stream is used for writing into the file (write operation). The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

* 1. **Threading**

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time-consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job. Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application. In C#, the **System.Threading.Thread** class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the main thread. When a C# program starts execution, the main thread is automatically created. The threads created using the Thread class are called the child threads of the main thread. You can access a thread using the **CurrentThread** property of the Thread class.

The life cycle of a thread starts when an object of the **System.Threading.Thread** class is created and ends when the thread is terminated or completes execution. Following are the various states in the life cycle of a thread.

* **The Unstarted State** − It is the situation when the instance of the thread is created but the Start method is not called.
* **The Ready State** − It is the situation when the thread is ready to run and waiting CPU cycle.
* **The Not Runnable State** − A thread is not executable, when
  + Sleep method has been called
  + Wait method has been called
  + Blocked by I/O operations
* **The Dead State** − It is the situation when the thread completes execution or is aborted.
  1. **CodeDom**

CodeDOM stands for Code Document Object Model. The CodeDOM provides types that represent many common types of source code elements. CodeDOM is the technique to generate code at run-time. As simplistic the definition seems, it is indeed one of the most powerful techniques in the repertoire of software developers. It is widely used in ORM - Object Relational Mapping. Besides dynamically mapping objects to scalar database entities, code generation can also be used to provide a plug-in framework in our applications which enables users to enhance features in the basic application. Code generation is an important instrument for programming methodologies like Generative Programming and Aspect Oriented Programming.

CodeDOM is a mechanism provided by the .NET Framework which lets us generate source code in multiple languages using a single model. We create code graphs and use the methods provided for CodeDOM to generate code in a language of our choice. Then we can use dynamic code compilation classes (also provided by CodeDOM) to generate assemblies which can then be loaded and used dynamically. To represent source code, CodeDOM elements are linked to each other to form a data structure known as a CodeDOM graph, which models the structure of some source code. The **System.CodeDom** namespace defines types that can represent the logical structure of source code, independent of a specific programming language. The **System.CodeDom.Compiler** namespace defines types for generating source code from CodeDOM graphs and managing the compilation of source code in supported languages.

* 1. **WMI**

Windows Management Instrumentation (WMI) is a clever way to give .NET developers remote and local access to information about their computer, services, and devices. WMI is a set of extensions Microsoft created to provide instrumentation (measurement) information from within Windows. However, WMI is a Microsoft-specific set of extensions that are actually built upon the WBEM (Web-Based Enterprise Management) and CIM (Common Information Model) standards developed by the DMTF (Distributed Management Task Force). The purpose of WMI is to define a proprietary set of environment-independent specifications which allow management information to be shared between management applications. WMI prescribes enterprise management standards and related technologies for Windows that work with existing management standards, such as Desktop Management Interface (DMI) and Simple Network Management Protocol (SNMP). WMI complements these other standards by providing a uniform model. This model represents the managed environment through which management data from any source can be accessed in a common way.

* 1. **Windows Service Applications**

In Windows NT operating systems, a Windows service is a computer program that operates in the background. A Windows service must conform to the interface rules and protocols of the Service Control Manager, the component responsible for managing Windows services. It is the Services and Controller app, services.exe, that launches all the services and manages their actions, such as start, end, and etc. Windows services can be configured to start when the operating system is started and run in the background as long as Windows is running. Alternatively, they can be started manually or by an event. Windows NT operating systems include numerous services which run in context of three user accounts: System, Network Service and Local Service. These Windows components are often associated with Host Process for Windows Services. Because Windows services operate in the context of their own dedicated user accounts, they can operate when a user is not logged on.

1. **Case Studies**
   1. **Developing Graphics Applications using GDI+**
   2. **Developing Network Applications**
   3. **Creating and Publishing Customizable Web Reports**
   4. **Database Application**
   5. **Web Service**