**SPIDER ALERT**

**ABSTRACT**

Social network worms, such as email worms and face book worms, pose a critical security threat to the Internet. Modeling their propagation dynamics is essential to predict their potential damages and develop countermeasures. Although several analytical models have been proposed for modeling propagation dynamics of social network worms, there are two critical problems unsolved: temporal dynamics and spatial dependence. First, previous models have not taken into account the different time periods of Internet users checking emails or social messages, namely, temporal dynamics. Second, the problem of spatial dependence results from the improper assumption that the states of neighboring nodes are independent. These two problems seriously affect the accuracy of the previous analytical models. To address these two problems, we propose a novel analytical model. This model implements a spatial temporal synchronization process, which is able to capture the temporal dynamics. Additionally, we find the essence of spatial dependence is the spreading cycles. By eliminating the effect of these cycles, our model overcomes the computational challenge of spatial dependence and provides a stronger approximation to the propagation dynamics. To evaluate our susceptible-infectious immunized (SII) model, we conduct both theoretical analysis and extensive simulations. Compared with previous epidemic models and the spatial-temporal model, the experimental results show our SII model achieves a greater accuracy. We also compare our model with the susceptible-infectious-susceptible and susceptible-infectious-recovered models. The results show that our model is more suitable for modeling the propagation of social network worms.

**Existing system**

The existing system Social networks have become attractive targets for worm creators because of the following characteristics. First, they rely on the information like contact lists contained in a victim’s machine to locate new targets. This intelligent mechanism allows far more efficient propagation than traditional scanning worms that make a large number of wild guesses for every successful infection. Second, by using social engineering techniques exploiting trust in social networks, many users fail to recognize malicious codes that are sent by their friends and subsequently become infected. This results in a wide range that worms propagate to. Third, researchers have found that social networks exhibit both small world properties and scale-free behaviors. This means that the spreading of social network worms can be incredibly fast because the highly connected “hub” nodes of a scale-free network and the short paths in a strongly clustered small world will greatly facilitate the propagation of an infection over the whole network.

**Disadvantages**

1. The virus spread of social network worms depends on human involvement. That is Internet users periodically check their newly arrived emails or messages, and are lured to open those which are actually worm copies.
2. The period for checking these emails or messages depends on users’ own patterns.
3. Previous models cannot present accurate and realistic spreading procedure.
4. The spreading of social network worms relies on the topology of social networks. In the modeling,
5. The probability of a node being infected will increase when its neighbors have been infected.

**Proposed system**

In this proposed system extensive analyses on two important problems, temporal dynamics and spatial dependence, which crucially affect the accuracy of the existing analytical models. Our analysis shows that previous works cannot model the realistic propagation with different checking time periods of users. Moreover, the analysis shows the spreading cycles formed in the modeling lead to considerable errors in estimating the infection probabilities. We propose a novel SII model. This model implements a spatial-temporal synchronization process, which helps us to solve the problem of modeling temporal dynamics. Furthermore, by eliminating the effect of those cycles, the SII model overcomes the computational challenge and provides a stronger approximation of spatial dependence.

**Advantages**

1. SII model is able to address two critical problems unsolved in the previous analytical models: temporal dynamics and spatial dependence.
2. Stronger approximation to the spatial dependence
3. SII model is accurate for modeling the spread of social network worms
4. SII model is more suitable for modeling the propagation of social network worms

**Modules**

1. Network analyzer
2. Node creation
3. Data transfer
4. Virus Propagation
5. Find affect nodes

**Network analyzer**

The network has divided by workgroups. This module will help us to get the connected and the active systems in the network. After getting login to our process, this module will get the connected systems and shows to the users. The user can select the system to deliver their data by file transfer.

In this module, each node registers the details such as Node IP address. Nodes details are stored and maintained in sever. After that Nodes enter the ip and to activate themselves in the network.

**Node creation**

In this module we have registered the node name and IP address this detail store the Database After click Login page enter the node name and sender decides to send data to receiver, it performs Data and sends the result to receiver, node creation id identification of data path.

**Data transfer**

The user has choice for browse the file to send. And it the data send to the receiver. By the receiver can gets the data will be sent by sender .And then the choice is authentication to produce it all.

Generally in network before transmission the data’s are converted into the network adaptable format, only these converted data are transmitted on the network. In this module the transmitted data are converted to data by using the coding and modulation technique.

**Virus Propagation**

In this module spreading of social network worms can be incredibly fast because the highly connected nodes of a scale-free network and the short paths in a strongly clustered small world will greatly facilitate the propagation of an infection over the whole network nodes.

In this module we drop the corrupted node by using the Cyclic Redundancy Check. Data packet is correct, the packet acknowledgment will be sent back to transmitter. Otherwise, the receiver drops the corrupted data packet and waits for the packet retransmission.

**Find affect nodes**

In this module SII model is find the spread of social network worm scan finds the virus and blocks the transaction from that abnormal file. SII is used for find Propagation. in this modules find the affected node from sender data. if virus data is detected node find the not send to receiver.

**CHAPTER 2**

**FEASIBILITY STUDY:**

A feasibility study is a preliminary study undertaken to determine and document a project's viability. The term feasibility study is also used to refer to the resulting document. These results of this study are used to make a decision whether to proceed with the project, or table it. If it indeed leads to a project being approved, it will - before the real work of the proposed project starts - be used to ascertain the likelihood of the project's success. It is an analysis of possible alternative solutions to a problem and a recommendation on the best alternative. It, for example, can decide whether an order processing be carried out by a new system more efficiently than the previous one.

A feasibility study could be used to test a proposal for new system, which could be used because:

* The current system may no longer carry its purpose,
* Technological advancement may have rendered the current system obsolete,
* The business is expanding, allowing it to cope with extra work load,
* Customers are complaining about the speed and quality of work the business provides,
* Competitors are now winning a big enough market share due to an effective integration of a computerized system.

A feasibility study should examine three main areas:

* Market issues
* Technical and organizational requirements
* Financial overview

Within a feasibility study, seven areas must be reviewed, including those of a :

* 1. Needs Analysis,
  2. Economics,
  3. Technical,
  4. Schedule,
  5. Organizational,
  6. Cultural, and
  7. Legal.

**Needs Analysis**

A needs analysis should be the first undertaking of a feasibility study as it clearly defines the project outline and the clients' requirements. Once these questions have been answered the person/s undertaking the feasibility study will have outlined the project needs definition. The following questions need to be asked to define the project needs definition: What is the end deliverable? What purpose will it serve? What are the environmental effects? What are the rules and regulations? What standards will we be measured against? What are the quality requirements? What is the minimal quality requirements allowed? What sustainability can we expect? What carry over work can we expect? What are the penalty clauses? How much do we need to outsource? How much do we need to in source?

**Technical feasibility study**

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on an outline design of system requirements in terms of Input, Output, Fields, Programs, and Procedures. This can be qualified in terms of volumes of data, trends, frequency of updating, etc... In order to give an introduction to the technical system.

**Cultural Feasibility study**

In this stage, the project's alternatives are evaluated for their impact on the local and general culture. For example, environmental factors need to be considered.

**Legal Feasibility study**

Not necessarily last, but all projects must face legal scrutiny. When an organization either has legal council on staff or on retainer, such reviews are typically standard. However, any project may face legal issues after completion too.

**Marketing Feasibility study**

'''''''This will include analysis of single and multi-dimensional market forces that could affect the commercial, along with the company that is carrying out the feasibility achieving more and more reputation as they have carried out safety checks which allow the system to run appropriately.''''

Advantages of making Feasibility study:

There are many advantages of making feasibility study some of which are summarized below:

* This study being made as the initial step of software development life cycle has all the analysis part in it which helps in analyzing the system requirements completely.   
  .
* Helps in identifying the risk factors involved in developing and deploying the system   
  .
* The feasibility study helps in planning for risk analysis   
  .
* Feasibility study helps in making cost/benefit analysis which helps the rganization and system to run efficiently.   
  .
* Feasibility study helps in making plans for training developers for implementing the system.   
  .
* So a feasibility study is a report which could be used by the senior or top persons in the organization. This is because based on the report the organization decides about cost estimation, funding and other important decisions which is very essential for an organization to run profitably and for the system to run stable.

**DEVELOPMENT ENVIRONMENT**

**SOFTWARE CONFIGURATION:**

* Front End : VISUAL STUDIO.NET 2008
* Code Behind : C#.NET
* Back End : SQL SERVER 2005
* Operating System : Windows 10

**HARDWARE CONFIGURATION:**

* Hard disk : 500 GB
* RAM : 2GB
* Processor : I3
* Monitor : 17” Color monitor
* Key board, Mouse : Multi media

**FEATURES OF THE SOFTWARE:**

**Introduction To .NET:**

.NET is the Microsoft’s development model in which software becomes available over the Internet. The .Net Framework is the infrastructure of .Net. .Net is built from the group up on open architecture. The goal of the Microsoft .Net platform is to simplify web development.

The .Net Framework provides the foundation upon which application and XML web services are build and executed the unified nature of the .Net Framework means that all applications, whether they are windows applications, web applications are XML web services are developer by using a common set tools and code, and are easily integrated with one another.

### The .Net Framework consists of the following elements:

* **The Common Language Runtime.** The runtime handles runtime services, including language integration, security and memory management. During development, the runtime provides features that are needed to simplify development.
* **Class Libraries.** Class libraries provide reusable code for most common task, including data access, XML web service development, and web and windows forms.

**Benefits of .Net Framework:**

The benefits of using the .Net Framework for developing application include:

* Based on Web standards and practices.
* Design using unified application models.
* Easy for developers to use.

### About VC#.Net:

Visual C# (pronounced C sharp) is designed to be a fast and easy way to create .Net applications, including Web services and ASP.NET Web applications. Applications written in Visual C# are built on the services of the common language runtime and take full advantage of the .Net Framework.

With its many innovations, C# enables rapid application development while retaining the expressiveness and elegance of C-style languages. Visual Studio supports Visual C# with a full-featured Code Editor, project templates, designers, code wizards, a powerful and easy-to-use debugger and other tools. The .Net Framework class library provides access to a wide range of operating system services and other useful, well-designed class that speed up the development cycle significantly.

C# is a simple, elegant, type-safe, object-oriented language recently developed by Microsoft for building a wide range of applications. C# is designed to bring rapid development to the C++ programmer without sacrificing the power and control that are a hallmark of C and C++.

Because of the heritage, C# has a high degree of fidelity with C and C++, and developers familiar with these languages can quickly become productive in C#.

C# provides intrinsic code trust mechanisms for a high level of security, garbage collection and type safety. C# supports single inheritance and creates Microsoft intermediate language (MSIL) as to native code compilers.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance and polymorphism. All variables and methods, including the Main method, the application’s entry point are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to avoid accidental redefinition.

### .Net Framework Architecture:

C# programs run on the .Net Framework, an integral component of Windows that includes a virtual execution System called the Common Language Runtime (CLR), an international standard that is the basis for creating and development environments in which languages and libraries work together seamlessly.

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code, along with resources such as bitmaps and strings, is stored on disk in an executable file called assembly, typically with an extension of .exe or .dll. An assembly contains a manifest that provides information on the assembly’s types, version, and culture and security requirements.

When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest. Then, if the security requirements are met, the CLR performs Just In Time (JIT) compilation to convert the IL code into native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling and resource management. Code that is executed by the CLR is sometimes referred to as “managed code” in contrast to “unmanaged code” which is compiled into native machine language that targets a specific system.

Language interoperability is a key feature of the .Net Framework. Because the IL code produced by the C# compiler conforms to the Common Type Specification (CTS), IL code generated from C# can interact with code that was generated from the .Net versions of Visual Basic, Visual C++, Visual J#, or any of more than 20 other CTS-compliant languages. A single assembly may contain multiple modules written in different .NET languages and the types can reference each other just as if they were written in the same language.

### FEATURES OF ASP.NET

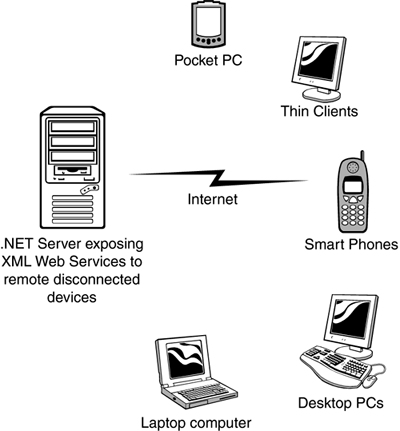
### What is .NET?

### When .NET was announced in late 1999, Microsoft positioned the technology as a platform for building and consuming Extensible Markup Language (XML) Web services. XML Web services allow any type of application, be it a Windows- or browser-based application running on any type of computer system, to consume data from any type of server over the Internet.

The reason this idea is so great is the way in which the XML messages are transferred: over established standard protocols that exist today. Using protocols such as SOAP, HTTP, and SMTP, XML Web services make it possible to expose data over the wire with little or no modifications to your existing code.

Figure presents a high-level overview of the .NET Framework and how XML Web services are positioned.

**Figure 1.1. Stateless XML Web services model.**

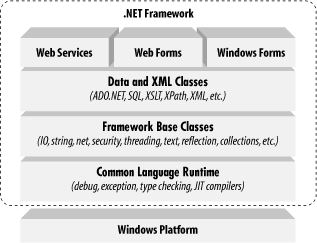


Since the initial announcement of the .NET Framework, it's taken on many new and different meanings to different people. To a developer, .NET means a great environment for creating robust distributed applications. To an IT manager, .NET means simpler deployment of applications to end users, tighter security, and simpler management. To a CTO or CIO, .NET means happier developers using state-of-the-art development technologies and a smaller bottom line.

To understand why all these statements are true, you need to get a grip on what the .NET Framework consists of, and how it's truly a revolutionary step forward for application architecture, development, and deployment.

### NET Framework

Now that you are familiar with the major goals of the .NET Framework, let's briefly examine its architecture. As you can see in Figure 1-2, the .NET Framework sits on top of the operating system, which can be a few different flavors of Windows and consists of a number of components .NET is essentially a system application that runs on Windows.



Conceptually, the CLR and the JVM are similar in that they are both runtime infrastructures that abstract the underlying platform differences. However, while the JVM officially supports only the Java language, the CLR supports any language that can be represented in its Common Intermediate Language (CIL). The JVM executes bytecode, so it can, in principle, support many languages, too. Unlike Java's bytecode, though, CIL is never interpreted. Another conceptual difference between the two infrastructures is that Java code runs on any platform with a JVM, whereas .NET code runs only on platforms that support the CLR. In April, 2003, the International Organization for Standardization and the International Electrotechnical Committee (ISO/IEC) recognized a functional subset of the CLR, known as the Common Language Interface (CLI), as an international standard.

This development, initiated by Microsoft and developed by ECMA International, a European standards organization, opens the way for third parties to implement their own versions of the CLR on other platforms, such as Linux or Mac OS X. For information on third-party and open source projects working to implement the ISO/IEC CLI and C# specifications

The layer on top of the CLR is a set of framework base classes. This set of classes is similar to the set of classes found in STL, MFC, ATL, or Java. These classes support rudimentary input and output functionality, string manipulation, security management, network communications, thread management, text management, reflection functionality, collections functionality, as well as other functions.

On top of the framework base classes is a set of classes that extend the base classes to support data management and XML manipulation. These classes, called ADO.NET, support persistent data management—data that is stored on backend databases. Alongside the data classes, the .NET Framework supports a number of classes to let you manipulate XML data and perform XML searching and XML translations.

Classes in three different technologies (including web services, Web Forms, and Windows Forms) extend the framework base classes and the data and XML classes. Web services include a number of classes that support the development of lightweight distributed components, which work even in the face of firewalls and NAT software. These components support plug-and-play across the Internet, because web services employ standard HTTP and SOAP.

Web Forms, the key technology behind ASP.NET, include a number of classes that allow you to rapidly develop web Graphical User Interface (GUI) applications. If you're currently developing web applications with Visual Interdev, you can think of Web Forms as a facility that allows you to develop web GUIs using the same drag-and-drop approach as if you were developing the GUIs in Visual Basic. Simply drag-and-drop controls onto your Web Form, double-click on a control, and write the code to respond to the associated event.

Windows Forms support a set of classes that allow you to develop native Windows GUI applications. You can think of these classes collectively as a much better version of the MFC in C++ because they support easier and more powerful GUI development and provide a common, consistent interface that can be used in all languages.

### The Common Language Runtime

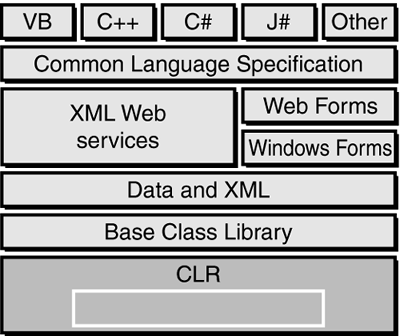
At the heart of the .NET Framework is the common language runtime. The common language runtime is responsible for providing the execution environment that code written in a .NET language runs under. The common language runtime can be compared to the Visual Basic 6 runtime, except that the common language runtime is designed to handle all .NET languages, not just one, as the Visual Basic 6 runtime did for Visual Basic 6. The following list describes some of the benefits the common language runtime gives you:

* **Automatic memory management**
* **Cross-language debugging**
* **Cross-language exception handling**
* **Full support for component versioning**
* **Access to legacy COM components**
* **XCOPY deployment**
* **Robust security model**

You might expect all those features, but this has never been possible using Microsoft development tools. Figure 1.3 shows where the common language runtime fits into the .NET Framework.

##### 

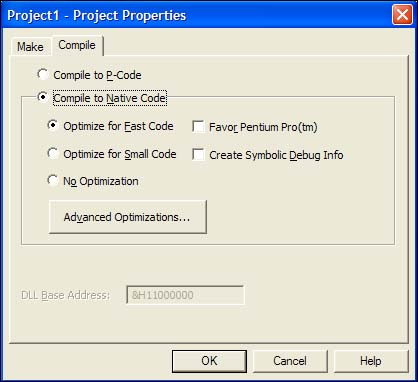
##### Figure 1.3. The common language runtime and the .NET Framework.



**Note** Code written using a .NET language is known as managed code. Code that uses anything but the common language runtime is known as unmanaged code. The common language runtime provides a managed execution environment for .NET code, whereas the individual runtimes of non-.NET languages provide an unmanaged execution environment.

**Inside the Common Language Runtime** The common language runtime enables code running in its execution environment to have features such as security, versioning, memory management and exception handling because of the way .NET code actually executes. When you compiled Visual Basic 6 forms applications, you had the ability to compile down to native node or p-code. Figure 1.4 should refresh your memory of what the Visual Basic 6 options dialog looked like.

**Figure** **1.4. Visual Basic 6 compiler options dialog.**



When you compile your applications in .NET, you aren't creating anything in native code. When you compile in .NET, you're converting your code—no matter what .NET language you're using—into an assembly made up of an intermediate language called Microsoft Intermediate Language (MSIL or just IL, for short). The IL contains all the information about your application, including methods, properties, events, types, exceptions, security objects, and so on, and it also includes metadata about what types in your code can or cannot be exposed to other applications. This was called a type library in Visual Basic 6 or an IDL (interface definition language) file in C++. In .NET, it's simply the metadata that the IL contains about your assembly.

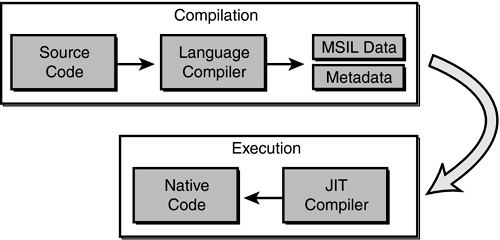
**Note** The file format for the IL is known as PE (portable executable) format, which is a standard format for processor-specific execution. When a user or another component executes your code, a process occurs called just-in-time (JIT) compilation, and it's at this point that the IL is converted into the specific machine language of the processor it's executing on. This makes it very easy to port a .NET application to any type of operating system on any type of processor because the IL is simply waiting to be consumed by a JIT compiler.

The first time an assembly is called in .NET, the JIT process occurs. Subsequent calls don't re-JIT the IL; the previously JITted IL remains in cache and is used over and over again. when you learn about Application Center Test, you also see how the warm-up time of the JIT process can affect application performance. Understanding the process of compilation in .NET is very important because it makes clear how features such as cross-language debugging and exception handling are possible. You're not actually compiling to any machine-specific code—you're simply compiling down to an intermediate language that's the same for all .NET languages. The IL produced by J# .NET and C# looks just like the IL created by the Visual Basic .NET compiler. These instructions are the same, only how you type them in Visual Studio .NET is different, and the power of the common language runtime is apparent.

When the IL code is JITted into machine-specific language, it does so on an as-needed basis. If your assembly is 10MB and the user is only using a fraction of that 10MB, only the required IL and its dependencies are compiled to machine language. This makes for a very efficient execution process. But during this execution, how does the common language runtime make sure that the IL is correct? Because the compiler for each language creates its own IL, there must be a process that makes sure what's compiling won't corrupt the system. The process that validates the IL is known as verification. Figure 1.5 demonstrates the process the IL goes through before the code actually executes.

##### 

##### Figure 1.5. The JIT process and verification.



When code is JIT compiled, the common language runtime checks to make sure that the IL is correct. The rules that the common language runtime uses for verification are set forth in the Common Language Specification (CLS) and the Common Type System (CTS).

### The .NET Framework Class Library

The second most important piece of the .NET Framework is the .NET Framework class library (FCL). As you've seen, the common language runtime handles the dirty work of actually running the code you write. But to write the code, you need a foundation of available classes to access the resources of the operating system, database server, or file server. The FCL is made up of a hierarchy of namespaces that expose classes, structures, interfaces, enumerations, and delegates that give you access to these resources.

The namespaces are logically defined by functionality. For example, the **System.Data** namespace contains all the functionality available to accessing databases. This namespace is further broken down into **System.Data.SqlClient**, which exposes functionality specific to SQL Server, and **System.Data.OleDb**, which exposes specific functionality for accessing OLEDB data sources. The bounds of a namespace aren't necessarily defined by specific assemblies within the FCL; rather, they're focused on functionality and logical grouping. In total, there are more than 20,000 classes in the FCL, all logically grouped in a hierarchical manner. Figure 1.8 shows where the FCL fits into the .NET Framework and the logical grouping of namespaces.

##### Figure 1.8. The .NET Framework class library.



To use an FCL class in your application, you use the Imports statement in Visual Basic .NET or the using statement in C#. When you reference a namespace in Visual Basic .NET or C#, you also get the convenience of auto-complete and auto-list members when you access the objects' types using Visual Studio .NET. This makes it very easy to determine what types are available for each class in the namespace you're using. As you'll see over the next several weeks, it's very easy to start coding in Visual Studio .NET.

**The Structure of a .NET Application**

To understand how the common language runtime manages code execution, you must examine the structure of a .NET application. The primary unit of a .NET application is the assembly. An assembly is a self-describing collection of code, resources, and metadata.

The assembly manifest contains information about what is contained within the assembly. The assembly manifest provides:

* Identity information, such as the assembly’s name and version number
* A list of all types exposed by the assembly
* A list of other assemblies required by the assembly
* A list of code access security instructions, including permissions required by the assembly and permissions to be denied the assembly

Each assembly has one and only one assembly manifest, and it contains all the description information for the assembly. However, the assembly manifest can be contained in its own file or within one of the assembly’s modules.

An assembly contains one or more modules. A module contains the code that makes up your application or library, and it contains metadata that describes that code. When you compile a project into an assembly, your code is converted from high-level code to IL. Because all managed code is first converted to IL code, applications written in different languages can easily interact. For example, one developer might write an application in Visual C# that accesses a DLL in Visual Basic .NET. Both resources will be converted to IL modules before being executed, thus avoiding any language-incompatibility issues.

Each module also contains a number of types. Types are templates that describe a set of data encapsulation and functionality. There are two kinds of types: reference types (classes) and value types (structures). These types are discussed in greater detail in Lesson 2 of this chapter. Each type is described to the common language runtime in the assembly manifest. A type can contain fields, properties, and methods, each of which should be related to a common functionality.

For example, you might have a class that represents a bank account. It contains fields, properties, and methods related to the functions needed to implement a bank account. A field represents storage of a particular type of data. One field might store the name of an account holder, for example.

Properties are similar to fields, but properties usually provide some kind of validation when data is set or retrieved. You might have a property that represents an account balance.

When an attempt is made to change the value, the property can check to see if the attempted change is greater than a predetermined limit. If the value is greater than the limit, the property does not allow the change. Methods represent behavior, such as actions taken on data stored within the class or changes to the user interface. Continuing with the bank account example, you might have a Transfer method that transfers a balance from a checking account to a savings account, or an Alert method that warns users when their balances fall below a predetermined level.

**Compilation and Execution of a .NET Application**

When you compile a .NET application, it is not compiled to binary machine code; rather, it is converted to IL. This is the form that your deployed application takes—one or more assemblies consisting of executable files and DLL files in IL form. At least one of these assemblies will contain an executable file that has been designated as the entry point for the application.

When execution of your program begins, the first assembly is loaded into memory. At this point, the common language runtime examines the assembly manifest and determines the requirements to run the program. It examines security permissions requested by the assembly and compares them with the system’s security policy. If the system’s security policy does not allow the requested permissions, the application will not run. If the application passes the system’s security policy, the common ­language runtime executes the code. It creates a process for the application to run in and begins application execution.

When execution starts, the first bit of code that needs to be executed is loaded into memory and compiled into native binary code from IL by the common language runtime’s Just-In-Time (JIT) compiler. Once compiled, the code is executed and stored in memory as native code. Thus, each portion of code is compiled only once when an application executes. Whenever program execution branches to code that has not yet run, the JIT compiler compiles it ahead of execution and stores it in memory as binary code. This way, application performance is maximized because only the parts of a program that are executed are compiled.

The .NET Framework base class library contains the base classes that provide many of the services and objects you need when writing your applications. The class library is organized into namespaces. A namespace is a logical grouping of types that perform related functions. For example, the System.Windows.Forms namespace contains all the types that make up Windows forms and the controls used in those forms.

Namespaces are logical groupings of related classes. The namespaces in the .NET base class library are organized hierarchically. The root of the .NET Framework is the System namespace. Other namespaces can be accessed with the period operator. A typical namespace construction appears as follows:

**System**

**System.Data**

**System.Data.SQLClient**

The first example refers to the System namespace. The second refers to the System.Data namespace. The third example refers to the System.Data.SQLClient namespace.

**Table 1.1 introduces some of the more commonly used .NET base class namespaces.**

| **Table 1-1. Representative .NET Namespaces** | |
| --- | --- |
| **Namespace** | **Description** |
| System | This namespace is the root for many of the low-level types required by the .NET Framework. It is the root for primitive data types as well, and it is the root for all the other namespaces in the .NET base class library. |
| System.Collections | This namespace contains classes that represent a variety of different container types, such as Array List, Sorted List, Queue, and Stack. You also can find abstract classes, such as Collection Base, which are useful for implementing your own collection functionality. |
| System.ComponentModel | This namespace contains classes involved in component creation and containment, such as attributes, type converters, and license providers. |
| System.Data | This namespace contains classes required for database access and manipulations, as well as additional namespaces used for data access. |
| System.Data.Common | This namespace contains a set of classes that are shared by the .NET managed data providers. |
| System.Data.OleDb | This namespace contains classes that make up the managed data provider for OLE DB data access. |
| System.Data.SQLClient | This namespace contains classes that are optimized for interacting with Microsoft SQL Server. |
| System.Drawing | This namespace exposes GDI+ functionality and provides classes that facilitate graphics rendering. |
| System.IO | In this namespace, you will find types for handling file system I/O. |
| System.Math | This namespace is home to common mathematics functions such as extracting roots and trigonometry. |
| System.Reflection | This namespace provides support for obtaining information and dynamic creation of types at runtime. |
| System.Security | This namespace is home to types dealing with permissions, cryptography, and code access security. |
| System.Threading | This namespace contains classes that facilitate the implementation of multithreaded applications. |
| System.Windows.Forms | This namespace contains types involved in creating standard Windows applications. Classes that represent forms and controls reside here as well. |

The namespace names are self-descriptive by design. Straightforward names make the .NET Framework easy to use and allow you to rapidly familiarize yourself with its contents.

**Introduction to Object-Oriented Programming**

Programming in the .NET Framework environment is done with objects. Objects are programmatic constructs that represent packages of related data and functionality. Objects are self-contained and expose specific functionality to the rest of the application environment without detailing the inner workings of the object itself. Objects are created from a template called a class. The .NET base class library provides a set of classes from which you can create objects in your applications. You also can use the Microsoft Visual Studio programming environment to create your own classes. This lesson introduces you to the concepts associated with object-oriented programming.

**Objects, Members, and Abstraction**

An object is a programmatic construct that represents something. In the real world, objects are cars, bicycles, laptop computers, and so on. Each of these items exposes specific functionality and has specific properties. In your application, an object might be a form, a control such as a button, a database connection, or any of a number of other constructs. Each object is a complete functional unit, and contains all of the data and exposes all of the functionality required to fulfill its purpose. The ability of programmatic objects to represent real-world objects is called abstraction.

**Classes Are Templates for Objects**

Classes were discussed in Chapter 1 and represent user-defined reference types. Classes can be thought of as blueprints for objects: they define all of the members of an object, define the behavior of an object, and set initial values for data when appropriate. When a class is instantiated, an in-memory instance of that class is created. This instance is called an object. To review, a class is instantiated using the New (new) keyword as follows:

Visual Basic .NET

' Declares a variable of the Widget type

Dim myWidget As Widget

' Instantiates a new Widget object and assigns it to the myWidget

' variable

myWidget = New Widget()

When an instance of a class is created, a copy of the instance data defined by that class is created in memory and assigned to the reference variable. Individual instances of a class are independent of one another and represent separate programmatic constructs. There is generally no limit to how many copies of a single class can be instantiated at any time. To use a real-world analogy, if a car is an object, the plans for the car are the class. The plans can be used to make any number of cars, and changes to a single car do not, for the most part, affect any other cars.

**Objects and Members**

Objects are composed of members. Members are properties, fields, methods, and events, and they represent the data and functionality that comprise the object. Fields and properties represent data members of an object. Methods are actions the object can perform, and events are notifications an object receives from or sends to other objects when activity happens in the application.

To continue with the real-world example of a car, consider that a Car object has fields and properties, such as Color, Make, Model, Age, Gas Level, and so on. These are the data that describe the state of the object. A Car object might also expose several methods, such as Accelerate, Shift Gears, or Turn. The methods represent behaviors the object can execute. And events represent notifications. For example, a Car object might receive an Engine Overheating event from its Engine object, or it might raise a Crash event when interacting with a Tree object.

**Object Models**

Simple objects might consist of only a few properties, methods, and perhaps an event or two. More complex objects might require numerous properties and methods and possibly even subordinate objects. Objects can contain and expose other objects as members. For example, the TextBox control exposes a Font property, which consists of a Font object. Similarly, every instance of the Form class contains and exposes a Controls collection that comprises all of the controls contained by the form. The object model defines the hierarchy of contained objects that form the structure of an object.

An object model is a hierarchical organization of subordinate objects contained and exposed within a main object. To illustrate, let’s revisit the example of a car as an object. A car is a single object, but it also consists of subordinate objects. A Car object might contain an Engine object, four Wheel objects, a Transmission object, and so on. The composition of these subordinate objects directly affects how the Car object functions as a whole.

For example, if the Cylinders property of the Engine subordinate object is equal to 4, the Car will behave differently than a Car whose Engine has a Cylinders property value of 8. Contained objects can have subordinate objects of their own. For example, the contained Engine object might contain several Sparkplug objects.

**Encapsulation**

Encapsulation is the concept that implementation of an object is independent of its interface. Put another way, an application interacts with an object through its interface, which consists of its public properties and methods. As long as this interface remains constant, the application can continue to interact with the component, even if implementation of the interface was completely rewritten between versions.

Objects should only interact with other objects through their public methods and properties. Thus, objects should contain all of the data they require, as well as all of the functionality that works with that data. The internal data of an object should never be exposed in the interface; thus, fields rarely should be Public (public).

Returning to the Car example. If a Car object interacts with a Driver object, the Car interface might consist of a GoForward method, a Go Backward method, and a Stop method. This is all the information that the Driver needs to interact with the Car. The Car might contain an Engine object, for example, but the Driver doesn’t need to know about the Engine object—the entire Driver cares about is that the methods can be called and that they return the appropriate values. Thus, if one Engine object is exchanged for another, it makes no difference to the Driver as long as the interface continues to function correctly.

**Polymorphism**

Polymorphism is the ability of different classes to provide different implementations of the same public interfaces. In other words, polymorphism allows methods and properties of an object to be called without regard for the particular implementation of those members. For example, a Driver object can interact with a Car object through the Car public interface.

If another object, such as a Truck object or a SportsCar object, exposes the same public interface, the Driver object can interact with them without regard to the specific implementation of that interface. There are two principal ways through which polymorphism can be provided: interface polymorphism and inheritance polymorphism.

**Interface Polymorphism**

An interface is a contract for behavior. Essentially, it defines the members a class should implement, but states nothing at all about the details of that implementation. An object can implement many different interfaces, and many diverse classes can implement the same interface. All objects implementing the same interface are capable of interacting with other objects through that interface. For example, the Car object in the previous examples might implement the IDrivable interface (by convention, interfaces usually begin with I), which specifies the GoForward, GoBackward, and Halt methods. Other classes, such as Truck, Forklift, or Boat might implement this interface and thus are able to interact with the Driver object. The Driver object is unaware of which interface implementation it is interacting with; it is only aware of the interface itself. Interface polymorphism is discussed in detail in Lesson 3.

**Inheritance Polymorphism**

Inheritance allows you to incorporate the functionality of a previously defined class into a new class and implement different members as needed. A class that inherits another class is said to derive from that class, or to inherit from that class. A class can directly inherit from only one class, which is called the base class. The new class has the same members as the base class, and additional members can be added as needed. Additionally, the implementation of base members can be changed in the new class by overriding the base class implementation.

Inherited classes retain all the characteristics of the base class and can interact with other objects as though they were instances of the base class. For example, if the Car class is the base class, a derived class might be Sports Car. The Sports Car class might be the base class for another derived class, the Convertible Sports Car. Each newly derived class might implement additional members, but the functionality defined in the original Car class is retained.

**Overview of ADO.NET**

Most applications require some kind of data access. Desktop applications need to integrate with central databases, Extensible Markup Language (XML) data stores, or local desktop databases. ADO.NET data-access technology allows simple, powerful data access while maximizing system resource usage.

Different applications have different requirements for data access. Whether your application simply displays the contents of a table, or processes and updates data to a central SQL server, ADO.NET provides the tools to implement data access easily and efficiently.

**Disconnected Database Access**

Previous data-access technologies provided continuously connected data access by default. In such a model, an application creates a connection to a database and keeps the connection open for the life of the application, or at least for the amount of time that data is required. However, as applications become more complex and databases serve more and more clients, connected data access is impractical for a variety of reasons, including the following:

* Open database connections are expensive in terms of system resources. The more open connections there are, the less efficient system performance becomes.
* Applications with connected data access are difficult to scale. An application that can comfortably maintain connections with two clients might do poorly with 10 and be completely unusable with 100.
* Open database connections can quickly consume all available database licenses, which can be a significant expense. In order to work within a limited set of client licenses, connections must be reused whenever possible.

ADO.NET addresses these issues by implementing a disconnected data access model by default. In this model, data connections are established and left open only long

enough to perform the requisite action. For example, if an application requests data from a database, the connection opens just long enough to load the data into the application, and then it closes. Likewise, if a database is updated, the connection opens to execute the UPDATE command, and then closes again. By keeping connections open only for the minimum required time, ADO.NET conserves system resources and allows data access to scale up with a minimal impact on performance.

**ADO.NET Data Architecture**

Data access in ADO.NET relies on two entities: the DataSet, which stores data on the local machine, and the Data Provider, a set of components that mediates interaction between the program and the database.

**The DataSet**

The DataSet is a disconnected, in-memory representation of data. It can be thought of as a local copy of the relevant portions of a database. Data can be loaded into a DataSet from any valid data source, such as a SQL Server database, a Microsoft Access database, or an XML file. The DataSet persists in memory, and the data therein can be manipulated and updated independent of the database. When appropriate, the DataSet can then act as a template for updating the central database.

The DataSet object contains a collection of zero or more DataTable objects, each of which is an in-memory representation of a single table. The structure of a particular DataTable is defined by the DataColumns collection, which enumerates the columns in a particular table, and the Constraint collection, which enumerates any constraints on the table. Together, these two collections make up the table schema. A DataTable also contains a DataRows collection, which contains the actual data in the DataSet.

The DataSet contains a DataRelations collection. A DataRelation object allows you to create associations between rows in one table and rows in another table. The DataRelations collection enumerates a set of DataRelation objects that define the relationships between tables in the DataSet. For example, consider a DataSet that contains two related tables: an Employees table and a Projects table. In the Employees table, each employee is represented only once and is identified by a unique EmployeeID field. In the Projects table, an employee in charge of a project is identified by the EmployeeID field, but can appear more than once if that employee is in charge of multiple projects. This is an example of a one-to-many relationship; you would use a DataRelation object to define this relationship. Additionally, a DataSet contains an Extended Properties collection, which is used to store custom information about the DataSet

**The Data Provider**

The link to the database is created and maintained by a data provider. A data provider is not a single component; rather it is a set of related components that work together to provide data in an efficient, performance-driven manner. The first version of the Microsoft .NET Framework shipped with two data providers: the SQL Server .NET Data Provider, designed specifically to work with SQL Server 7 or later, and the OleDb .NET Data Provider, which connects with other types of databases. Microsoft Visual Studio .NET 2003 added two more data providers: the ODBC Data Provider and the Oracle Data Provider. Each data provider consists of versions of the following generic component classes:

* The Connection object provides the connection to the database.
* The Command object executes a command against a data source. It can execute non-query commands, such as INSERT, UPDATE, or DELETE, or return a DataReader with the results of a SELECT command.
* The DataReader object provides a forward-only, read-only, connected recordset.
* The DataAdapter object populates a disconnected DataSet or DataTable with data and performs updates.

Data access in ADO.NET is facilitated as follows: a Connection object establishes a connection between the application and the database. This connection can be accessed directly by a Command object or by a DataAdapter object.

The Command object provides direct execution of a command to the database. If the command returns more than a single value, the Command object returns a DataReader to provide the data. This data can be directly processed by application logic. Alternatively, you can use the DataAdapter to fill a DataSet object. Updates to the database can be achieved through the Command object or through the DataAdapter. The generic classes that make up the data providers are summarized in the following sections.

**The Connection Object**

The Connection object represents the actual connection to the database. Visual Studio .NET 2003 supplies two types of Connection classes: the SqlConnection object, which is designed specifically to connect to SQL Server 7 or later, and the OleDbConnection object, which can provide connections to a wide range of database types.

Visual Studio .NET 2003 further provides a multipurpose ODBCConnection class, as well as an Oracle Connection class optimized for connecting to Oracle databases. The Connection object contains all of the information required to open a channel to the database in the ConnectionString property. The Connection object also incorporates methods that facilitate data transactions.

**The Command Object**

The Command object is represented by two corresponding classes, SqlCommand and OleDbCommand. You can use Command objects to execute commands to a database across a data connection. Command objects can be used to execute stored procedures on the database and SQL commands, or return complete tables.

Command objects provide three methods that are used to execute commands on the database:

* ExecuteNonQuery.
  + Executes commands that return no records, such as INSERT, UPDATE, or DELETE
* ExecuteScalar.
  + Returns a single value from a database query
* ExecuteReader.
  + Returns a result set by way of a DataReader object

**The DataReader Object**

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of a data provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of a Command object’s ExecuteReader method. The SqlCommand.Execute­Reader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object.

Likewise, the ODBC and Oracle Command.ExecuteReader methods return a DataReader specific to the ODBC and Oracle Data Providers respectively. The DataReader can supply rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance, but it requires exclusive use of an open Connection object for the lifetime of the DataReader.

**The DataAdapter Object**

The DataAdapter is the class at the core of ADO.NET disconnected data access. It is essentially the middleman, facilitating all communication between the database and a DataSet. The DataAdapter fills a DataTable or DataSet with data from the database whenever the Fill method is called. After the memory-resident data has been manipulated, the DataAdapter can transmit changes to the database by calling the Update method.

The DataAdapter provides four properties that represent database commands. The four properties are:

* **Select Command.**

Contains the command text or object that selects the data from the database. This command is executed when the Fill method is called and fills a DataTable or a DataSet.

* **Insert Command.**

Contains the command text or object that inserts a row into a table.

* **Delete Command.**

Contains the command text or object that deletes a row from a table.

* **Update Command.**

Contains the command text or object that updates the values of a database.

When the Update method is called, changes in the DataSet are copied back to the database, and the appropriate Insert Command, Delete Command, or Update Command is executed.

**Accessing Data**

Visual Studio .NET has many built-in wizards and designers to help you shape your data-access architecture rapidly and efficiently. With minimal actual coding, you can implement robust data access for your application. However, the ADO.NET object model is fully available through code to implement customized features or to fine-tune your program. In this lesson, you will learn how to connect to a database with ADO.NET and retrieve data to your application. You will learn to use the visual designers provided by Visual Studio .NET and direct code access.

**Microsoft Visual Basic.Net**

With its release for the .NET platform, the Visual Basic language has undergone dramatic changes.

For example:

* + The language itself is now fully object-oriented.
  + Applications and components written in Visual Basic .NET have full access to the .NET Framework, an extensive class library that provides system and application services.
  + All applications developed using Visual Basic .NET run within a managed runtime environment, the .NET common language runtime.

Visual Basic .NET is the next generation of Visual Basic, but it is also a significant departure from previous generations. Experienced Visual Basic 6 developers will feel comfortable with Visual Basic .NET code and will recognize most of its constructs. However, Microsoft has made some changes to make Visual Basic .NET a better language and an equal player in the .NET world.

These include such additions as a Class keyword for defining classes and an Inherits keyword for object inheritance, among others. Visual Basic 6 code can't be compiled by the Visual Basic .NET compiler without significant modification. The good news is that Microsoft has provided a migration tool to handle the task.

**FEATURES OF SQL SERVER 2005**

Microsoft SQL server lets you quickly build powerful and reliable database applications. SQL server 7.0 highly scalable, fully relational, high performance, multi-user database server. That can be used by enterprise of any size to manage large amount of data for client\server applications.

The major new and improved features of SQL server 7.0 include the multi-user support Multi platform support, added memory support, scalability, integration with MMC, Microsoft Management console and improved multiple server management. Parallel database backup and restore. Data replication, Data warehousing distributed queries, distributed transactions, Dynamic cocking Internet Access, Integrated windows security, Mail integration Microsoft English Query, ODBC Support.

SQL Server management is accomplished through a set of component applications. SQL Server introduces a number of new and improved management tools that are SQL Server Enterprise management, profiles, and Query Analyzer service manager wizards.

The OLAP Services feature available in SQL Server version 7.0 is now called SQL Server 2000 Analysis Services. The term OLAP Services has been replaced with the term Analysis Services. Analysis Services also includes a new data mining component. The Repository component available in SQL Server version 7.0 is now called Microsoft SQL Server 2000 Meta Data Services. References to the component now use the term Meta Data Services. The term repository is used only in reference to the repository engine within Meta Data Services

SQL-SERVER database consist of six type of objects,

They are,

**1. TABLE**

**2. QUERY**

**3. FORM**

**4. REPORT**

**5. MACRO**

**TABLE:**

A database is a collection of data about a specific topic.

**VIEWS OF TABLE:**

We can work with a table in two types,

1. Design View

2. Datasheet View

**Design View**

To build or modify the structure of a table we work in the table design view. We can specify what kind of data will be hold.

**Datasheet View**

To add, edit or analyses the data itself we work in tables datasheet view mode.

**QUERY:**

A query is a question that has to be asked the data. Access gathers data that answers the question from one or more table. The data that make up the answer is either dynaset (if you edit it) or a snapshot(it cannot be edited).Each time we run query, we get latest information in the dynaset.Access either displays the dynaset or snapshot for us to view or perform an action on it ,such as deleting or updating.

**FORMS:**

A form is used to view and edit information in the database record by record .A form displays only the information we want to see in the way we want to see it. Forms use the familiar controls such as textboxes and checkboxes. This makes viewing and entering data easy.

**Views of Form:**

We can work with forms in several primarily there are two views,

They are,

1. Design View

2. Form View

**Design View**

To build or modify the structure of a form, we work in forms design view. We can add control to the form that are bound to fields in a table or query, includes textboxes, option buttons, graphs and pictures.

**Form View**

The form view which display the whole design of the form.

**REPORT:**

A report is used to vies and print information from the database. The report can ground records into many levels and compute totals and average by checking values from many records at once. Also the report is attractive and distinctive because we have control over the size and appearance of it.

**MACRO:**

A macro is a set of actions. Each action in macros does something. Such as opening a form or printing a report .We write macros to automate the common tasks the work easy and save the time.

**SYSTEM DESIGN**

INPUT DESIGN

Input design is the process of converting user-originated inputs to a computer-based format to the application forms. Input design is one of the most expensive phases of the operation of computerized system and is often the major problem of a system.

Input design forms are:

* **Login**
* **Send file**

**OUTPUT DESIGN**

Output design generally refers to the results and information that are generated by the system for many end-users; output is the main reason for developing the system and the basis on which they evaluate the usefulness of the application. The output is designed in such a way that it is attractive, convenient and informative. Forms are designed in C#.NET with various features, which make the console output more pleasing.

As the outputs are the most important sources of information to the users, better design should improve the system’s relationships with us and also will help in decision-making. Form design elaborates the way output is presented and the layout available for capturing information.

Output design forms are:

* **Received files**
* **virus**

DATABASE DESIGN

The database design is a must for any application developed especially more for the data store projects. Since the chatting method involves storing the message in the table and produced to the sender and receiver, proper handling of the table is a must. In the project, login table is designed to be unique in accepting the username and the length of the username and password should be greater than zero. Both the company and seeker username are stored in the same table with different flag values.

The job and question table is common to all companies. Likewise job apply details are stored in the common ‘apply’ table. The different users view the data in different format according to the privileges given. The complete listing of the tables and their fields are provided in the annexure under the title ‘Table Structure’.

**CHAPTER 5**

**TESTING AND IMPLEMENTATION**

**System Implementation:**

Implementation is the most crucial stage in achieving a successful system and giving the user’s confidence that the new system is workable and effective. Implementation of a modified application to replace an existing one. This type of conversation is relatively easy to handle, provide there are no major changes in the system.

Each program is tested individually at the time of development using the data and has verified that this program linked together in the way specified in the programs specification, the computer system and its environment is tested to the satisfaction of the user. The system that has been developed is accepted and proved to be satisfactory for the user. And so the system is going to be implemented very soon. A simple operating procedure is included so that the user can understand the different functions clearly and quickly.

Initially as a first step the executable form of the application is to be created and loaded in the common server machine which is accessible to all the user and the server is to be connected to a network. The final stage is to document the entire system which provides components and the operating procedures of the system.

**Testing:**

**System Testing:**

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. The final step involves Validation testing, which determines whether the software function as the user expected. The end-user rather than the system developer conduct this test most software developers as a process called “Alpha and Beta test” to uncover that only the end user seems able to find.

This is the final step in system life cycle. Here we implement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion. Here system maintenance is done every months or year based on company policies, and is checked for errors like runtime errors, long run errors and other maintenances like table verification and reports.

**Unit Testing:**

Unit testing verification efforts on the smallest unit of software design, module. This is known as “Module Testing”. The modules are tested separately. This testing is carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

**Integration Testing:**

Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole. In the integration-testing step, all the error uncovered is corrected for the next testing steps.

**Validation Testing:**

To uncover functional errors, that is, to check whether functional characteristics confirm to specification or not specified.

**CHAPTER 6**

**SYSTEM MAINTENANCE**

The objectives of this maintenance work are to make sure that the system gets into work all time without any bug. Provision must be for environmental changes which may affect the computer or software system. This is called the maintenance of the system. Nowadays there is the rapid change in the software world. Due to this rapid change, the system should be capable of adapting these changes. In our project the process can be added without affecting other parts of the system.

Maintenance plays a vital role. The system liable to accept any modification after its implementation. This system has been designed to favor all new changes. Doing this will not affect the system’s performance or its accuracy.

In the project system testing is made as follows:

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. Then the web form level testing is made. For example storage of data to the table in the correct manner.

In the form, the zero length username and password are given and checked. Also the duplicate username is given and checked. The client side validations are made.

The dates are entered in wrong manner and checked. Wrong email-id is given and checked.

This is the final step in system life cycle. Here we implement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion. Here system maintenance is done every months or year based on company policies, and is checked for errors like runtime errors, long run errors and other maintenances like table verification and reports.

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it’s constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

Implementation is the process of converting a new system design into operation. It is the phase that focuses on user training, site preparation and file conversion for installing a candidate system. The important factor that should be considered here is that the conversion should not disrupt the functioning of the organization.

**SYSTEM MAINTENANCE**

**Maintenance**

The term “Software Maintenance” is used to describe software engineering activities. Maintenance activities involve making enhancements to software products, adapting to new environments and correcting problems. Software product enhancements may involve providing new functional capabilities, improving user displays and nodes of interaction, upgrading external documents and internal documentation or upgrading the performance characteristics of a system. Adaptation of software to a new environment may involve moving the software to a different machine, or for instance, modifying the software to accommodate a new telecommunication protocol or an additional disk drives. Problem correction involves modification and revalidation of software to correct errors.

Many activities performed during software development enhance the maintainability of a software product. They are:-

**Analysis activities:**

The analysis phase of software development is concerned with determining customer requirements and constraints and establishing feasibility of the product.

* Develop standards and guidelines
* Set milestones for supporting documents
* Specify quality assurance procedures
* Identify likely product enhancements
* Determine resources required for maintenance
* Estimate maintenance costs

**Architectural Design Activities:**

* Emphasize clarity and modularity as design criteria
* Design to ease likely enhancement
* Use standardized notations to document, data flows, functions, structure and interconnections
* Observe the principles of information hiding, data abstraction and top-down hierarchical decomposition

**Detailed Design Activities**

* Use standardized notations to specify algorithms, data structures and procedure interface specifications
* Specify side effects and exception handling for each routine

**Implementation activities**

* Use single entry, single exit constructs
* Use standard indentation of constructs
* Use simple, clear coding style
* Use symbolic constants to parameterize routines
* Provide margins on resources
* Provide standard documentation
* Follow standard internal commenting guidelines

**Other activities:**

* Develop a maintenance guide
* Develop a test suite
* Provide test suite documentation

**CONCLUSION**

It is concluded that the application works well and satisfy the end users. The application is tested very well and errors are properly debugged. The application is simultaneously accessed from more than one system. Simultaneous login from more than one place is tested.

This system is user friendly so everyone can use easily. Proper documentation is provided. The end user can easily understand how the whole system is implemented by going through the documentation. The system is tested, implemented and the performance is found to be satisfactory. All necessary output is generated. Thus, the project is completed successfully.

Further enhancements can be made to the application, so that the application functions very attractive and useful manner than the present one. The speed of the transactions become more enough now.

**FUTURE AND ENHANCEMENT**

There is scope for future development of this project. The world of computer fields is not static; it is always subject to be dynamic. The technology which is famous today becomes outdated the very next day. To keep abstract of technical improvements, the system may be further refined. So, it is not concluded. Yet it will improve with further enhancements.

Enhancements can be done in an efficient manner. We can even update the same with further modification establishment and can be integrated with minimal modification. Thus the project is flexible and can be enhanced at anytime with more advanced features.

**BIBLIOGRAPHY**

Microsoft C#.NET Programmer’s Cook Book:

**-MATTHEW MACDONALD**

**(Tata McGraw Hill Edition)**

Microsoft Visual Basic.NET Programmer’s CookBook:

**-MATTHEW MACDONALD**

**(Tata McGrawHill Edition)**

Grey Buczek, **.NET developers guide 2002, Prentice-Hall India.**

Benolt Marchal, **C#NET by example 2003 – TataMcGraw- Hill.**

System Analysis & Design – **Alenis Leon.**

An Integral approach to software engineering – **Pankaj Jalole.**

***Online Reference***

* www.dotnetspider.com
* www.programersheaven.com
* www.sql-server-performance.com
* www.developerfusion.com
* www.winsocketdotnetworkprogramming.com

Data Flow Diagram

Database

Sender

Node monitoring

Monitoring

Virus detected

no virus

Affected Node

Receiver

ER Diagram

File Details

Send data

System info

Tables

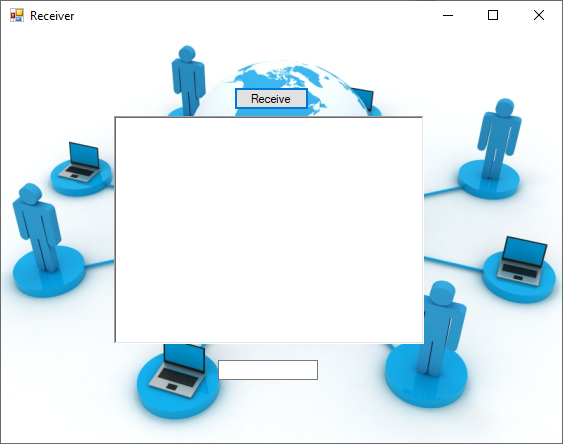
file details

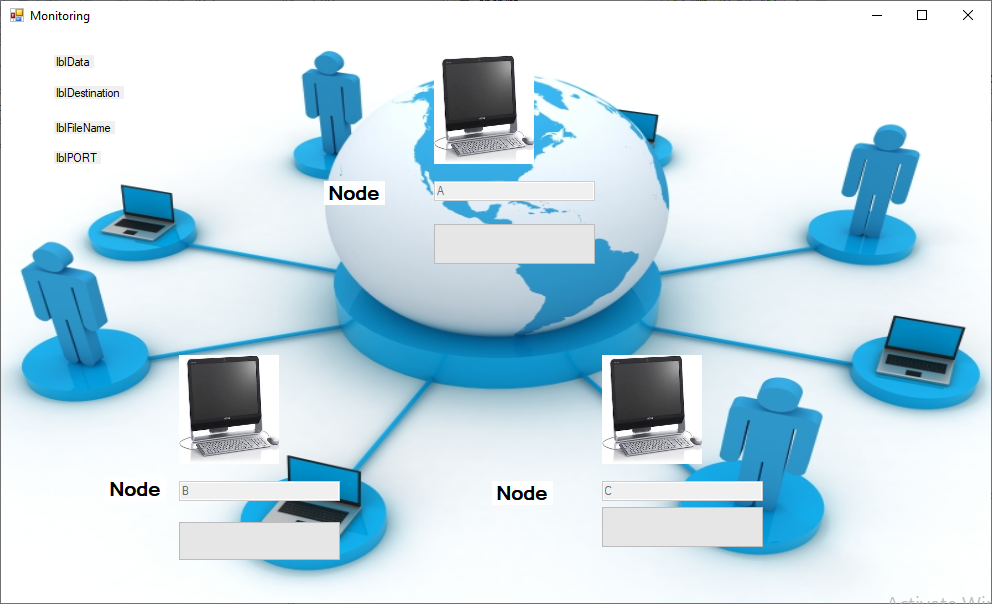
|  |  |  |
| --- | --- | --- |
| **Filed** | **Description** | **Data type** |
| Fid | Int | File id |
| Fname | Varchar | File name |
| Date | Date/timr | Date of transaction |
| Ip | Varchar | Ip address |

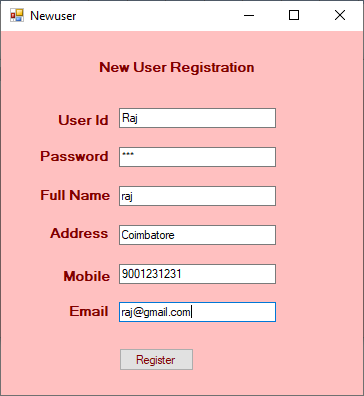
system info

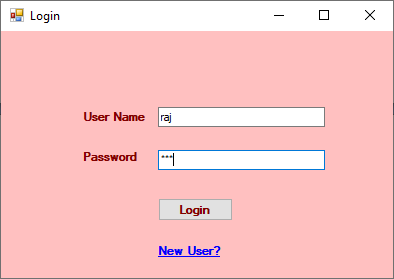
|  |  |  |
| --- | --- | --- |
| **Filed** | **Description** | **Data type** |
| Node | Int | node id |
| Ip | Varchar | Ip address |
| Port | Varchar | Port numer |

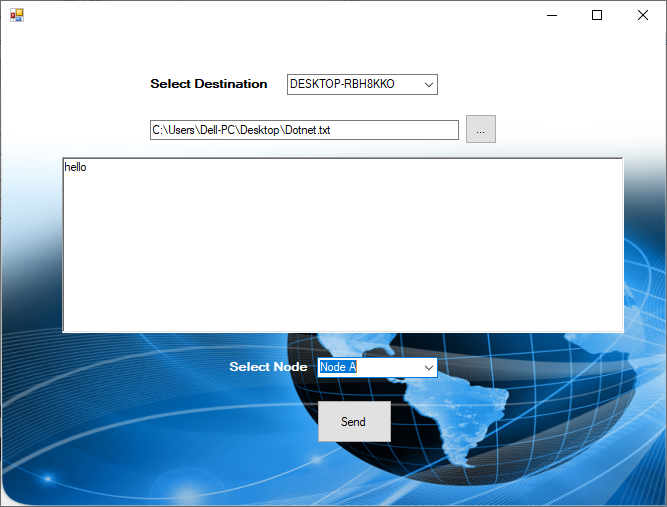
Screens

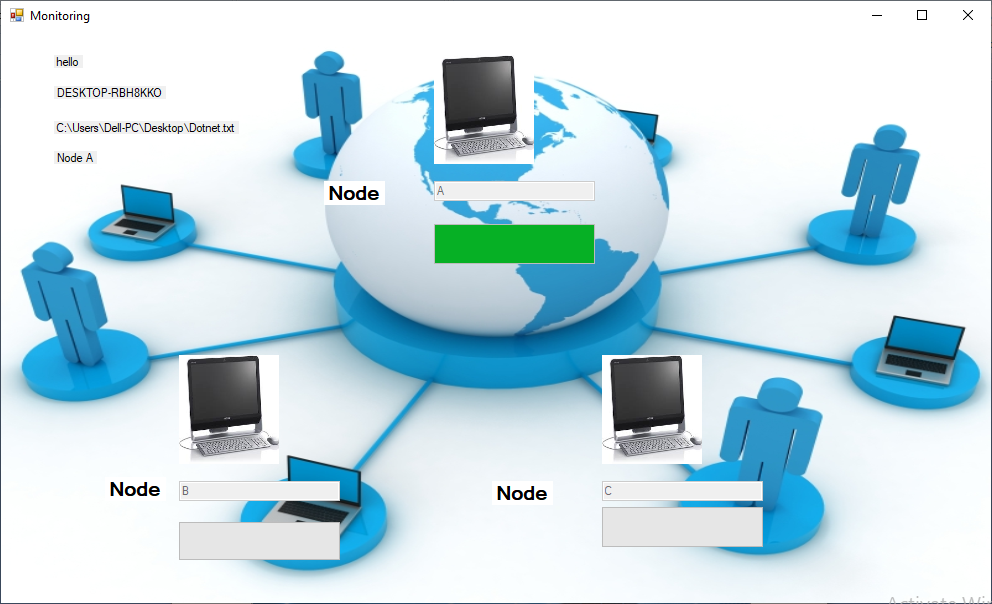


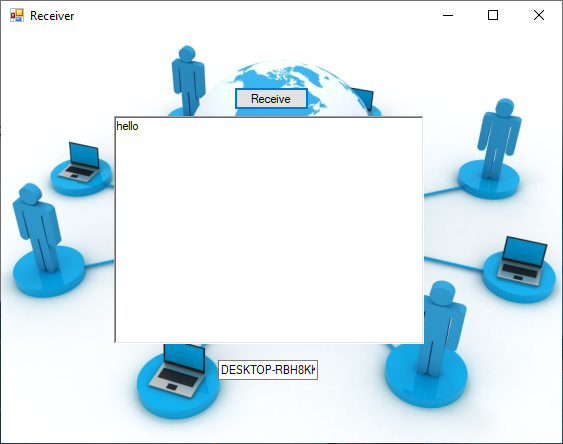


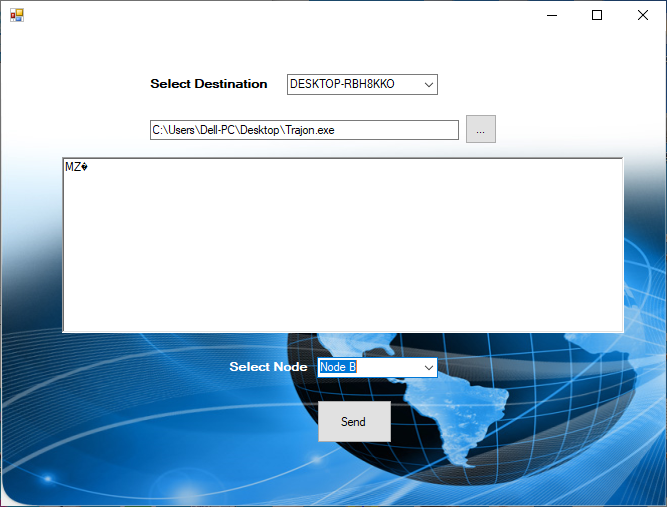


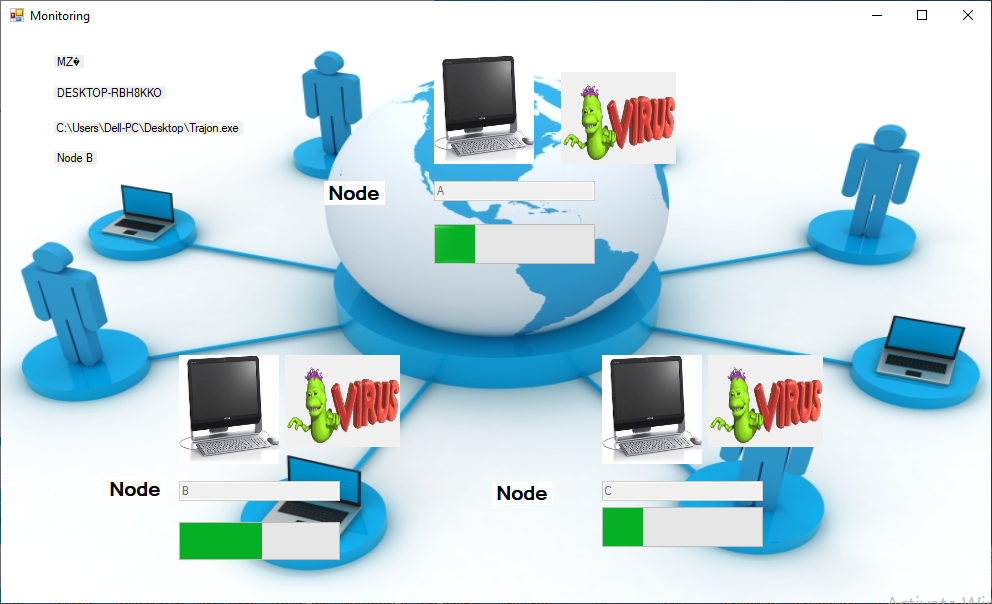












Code

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.Runtime.Serialization.Formatters.Binary;

using System.Security.Cryptography;

using System.IO;

namespace Receiver

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

MessageBox.Show("Ready To Receive Data..");

backgroundWorker1.RunWorkerAsync();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

public void ReceiveEnrouteDetails()

{

TcpListener list = new TcpListener(IPAddress.Any, 2000);

list.Start();

Socket soc = list.AcceptSocket();

string remote = soc.RemoteEndPoint.ToString();

NetworkStream ns = new NetworkStream(soc);

BinaryFormatter bf = new BinaryFormatter();

object source = bf.Deserialize(ns);

object message = bf.Deserialize(ns);

Control.CheckForIllegalCrossThreadCalls = false;

textBox1.Text = source.ToString();

richTextBox1.Text = message.ToString();

list.Stop();

soc.Close();

ns.Close();

if (soc.Connected == false)

{

ReceiveEnrouteDetails();

}

}

private void backgroundWorker1\_DoWork(object sender, DoWorkEventArgs e)

{

ReceiveEnrouteDetails();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace sender

{

public partial class Form2 : Form

{

SqlConnection con = new SqlConnection("server=.\\sqlexpress;Database=Virusspred;Integrated Security=true;");

SqlCommand cmd = new SqlCommand();

public Form2()

{

InitializeComponent();

}

private void Form2\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

if (textBox1.Text == "Admin" && textBox2.Text == "Admin")

{

//this.Close();

Adminhome fadm = new Adminhome();

fadm.Show();

}

else

{

cmd = new SqlCommand("select \* from Usertbl where Uid='" + textBox1.Text + "' and Pwd='" + textBox2.Text + "'", con);

con.Close();

con.Open();

SqlDataReader dr = cmd.ExecuteReader();

if (dr.Read())

{

Form1 fusr = new Form1();

fusr.Show();

}

else

{

MessageBox.Show("Login faild");

}

}

}

private void linkLabel1\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

Newuser ob = new Newuser();

ob.Show();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient;

namespace sender

{

public partial class Viewuser : Form

{

SqlConnection con = new SqlConnection("server=.\\sqlexpress;Database=Virusspred;Integrated Security=true;");

SqlCommand cmd = new SqlCommand();

public Viewuser()

{

InitializeComponent();

}

private void Viewuser\_Load(object sender, EventArgs e)

{

cmd = new SqlCommand("select \* from Usertbl", con);

con.Open();

SqlDataAdapter da = new SqlDataAdapter(cmd);

DataSet ds = new DataSet();

da.Fill(ds, "Usertbl");

dataGridView1.DataSource = ds.Tables[0].DefaultView;

con.Close();

}

}

}