CHAPTER 4

INTRODUCTION TO DEVELOPMENTAL TOOLS USED IN REAL TIME WEB-CENTRIC DIABETES DIAGNOSIS TELEMEDICINE SYSTEM

4.1 INTRODUCTION TO EXPERT SYSTEM

Expert systems, also called knowledge-based systems or knowledge systems, are computer systems characterized by the fact that an explicit distinction is made between a part in which knowledge of a problem domain is represented, and a part which manipulates that knowledge to reason about or solve an actual problem using problem data. Both the type of knowledge are used in solving the problem and the nature of the problem-solving methods used determine which problems can be solved. The knowledge represented in a knowledge base is formal in nature, and is the result of modeling essential features of the domain for the problem at hand. The incorporated knowledge may be acquired from domain experts, literature or datasets. Designing an expert system usually involves the use of methodologies for knowledge acquisition, modeling and evaluation.

The phrase knowledge-based system, or knowledge system, is generally employed to denote information systems in which some symbolic representation of human knowledge of a domain is applied,
usually to some extent resembling human reasoning, to solve actual problems in the domain.

Examples of problem domains include trouble shooting of equipment, medical diagnosis, financial advice, product design and so on. As this knowledge is often derived from experts in a particular field, and early knowledge-based systems were actually developed in close collaboration with experts, the term expert system was the term used in the early days to refer to these systems.

Knowledge, however, can also be extracted from literature or from datasets by using machine-learning methods. Moreover, not all domains of specific expert systems may be viewed as specialists’ fields. As a consequence, some people prefer to make a distinction between expert systems and knowledge-based systems. In their view the latter are more general than the former as the former should always concern a specialist's field. In this chapter, such a distinction will not be made as the techniques used in knowledge-based systems and the ones used in building expert systems are identical. Hence, the terms ‘expert system' and 'knowledge-based system' will be used inter changingly.

The present generation expert systems are capable of dealing with restricted problem domains. Collecting, maintaining and updating the incorporated knowledge taking into account its associated context, such as working environment, organization and field of expertise belongs to an area referred to as knowledge management.
The art of developing an expert system is called knowledge engineering, where there is emphasis on the pragmatic engineering aspects, or knowledge modeling, when development of domain models is emphasized. The process of collecting and analyzing knowledge in a problem domain is called knowledge acquisition or knowledge elicitation when the knowledge is gathered from interviews with experts, normally using interview techniques as developed by psychologists.

4.1.1 Expert system principles

As an expert system is a software system, the structure of expert systems can be described and understood in terms of the components of which such systems consist, as well as in terms of the interchange of information between these components. This is called an architecture.

- **Expert system architecture**

In the early years, expert systems were written in a high-level programming language, usually in LISP. However, using of such a programming language as an expert system building tool, demands disproportionate attention not proportionate to the implementational aspects of the system unrelated to the problem domain. Moreover, the expert knowledge of the domain and the algorithms for applying this knowledge will become densely interwoven.

This leads to systems that, once constructed, are practically not adaptable to changing views about the domain of concern. Expert knowledge, however, is dynamic: knowledge and experience are
continuously changing, requiring modifications of the related expert system. Attempts to solve this problem lead to the view that the domain knowledge and algorithms for applying this knowledge should be separated explicitly.

This principle constitutes the paradigm of today's expert system design:

\[
\text{expert system} = \text{knowledge} + \text{problem-solving methods}
\]

Accordingly, today's expert systems typically have two basic components:

A knowledge base that captures the domain specific knowledge, and an inference engine that consists of algorithms for manipulating the knowledge represented in the knowledge base to solve a problem presented to the system. In addition, an expert system may contain facilities to explain, illustrate or offer documentation for its reasoning steps, often called explanation facilities.

During the development of a system, it might be worthwhile to trace the reasoning behavior in more detail, which is provided by a trace facility. The capabilities of an inference engine are typically used to implement specific problem-solving methods, for example, methods to solve diagnostic problems.

Modern expert systems are rarely written in a high level programming language. Instead, they are built in a special software
environment, known under various names like expert system shells, expert-system builder tools, or knowledge based system toolkit.

An early example of such an environment is EMYCIN (Essential MYCIN), a system that emerged from MYCIN by stripping it of its knowledge concerning infectious disease. Other, more recent examples, include CLIPS, JESS, AION-DS. Also, the PROLOG programming language is eminently suitable to implement expert systems. Every expert system shell or builder tool offers a formal language, called a knowledge representation formalism, for encoding the domain knowledge in the knowledge base.

Furthermore, they provide one or more inference engines that are capable of manipulating knowledge that is represented in the formalism. The developer of an expert system is therefore shielded from most of the system's algorithmic aspects; only the domain-specific knowledge has to be provided and expressed in the knowledge-representation formalism, whereas the reasoning as offered by the tool may need to be tailored to the type of problem solving required. We observe that several advantages arise from the property that a knowledge base can be developed separately from the inference engine. A knowledge base can be developed and refined stepwise.

Errors and inadequacies can be easily remedied without making any changes to the program text. Furthermore, an expert-system builder tool can be used to develop expert systems for different problem domains, which may save in development time and costs.
• **Problem-solving methods**

As stated, the inference engine of an expert system shell is normally customized to obtain more specific problem-solving methods. An example is a diagnostic method that is able to utilize causal knowledge about the relationship between causes and the associated effects to explain observed malfunction of a device in terms of possible causes of that malfunction. Sometimes, the same problem can be solved in different ways, using different types of knowledge and different methods.

For example, the faults of a device can also be diagnosed by using expert knowledge saying that a particular combination of findings is typical for the occurrence of a particular fault. In this case, so-called heuristic associations rather than causal knowledge is used to diagnose malfunction. More about this will be said later wherein the formal properties of some methods will be examined. Typical examples of problems for which specific methods have been developed are:

• Diagnosis
• Prediction
• Planning and scheduling
• Design
• Decision making
Knowledge representation and inference

Key issue for the suitability of any expert system builder tool are the features it offers to model particular problem domains. In particular, the knowledge-representation formalism and the types of reasoning supported are of major importance. Logic, probability theory and decision theory are sufficiently general to permit describing the nature of knowledge representation, inference and problem solving without having to resort to special-purpose languages.

4.2 INTRODUCTION TO .NET

Contrary to the general belief, .NET is not a single technology. Rather, it is a set of technologies that work together seamlessly to solve our business problems.

Applications that can be developed by dot net :

✓ .NET provides a solid foundation for developing Internet or networked applications.
✓ ASP.NET Web applications: These include dynamic and data driven browser based applications.
✓ Windows form based applications: These refer to traditionally rich client applications.
✓ Console applications: These refer to traditional DOS kind of applications like batch scripts.
✓ Component libraries: These refer to components that typically encapsulate some business logic.
✓ Windows custom controls: As with traditional ActiveX controls, windows controls can be developed.

✓ Web custom controls: The concept of custom controls can be extended to Web applications allowing code reuse and modularization.

✓ Web services: They are “Web callable” functionality available via industry standards like HTTP, XML and SOAP.

✓ Windows services: They refer to applications that run as services in the background. They can be configured to start automatically when the system boots up.

- **.NET Framework SDK**

.NET has a complete Software Development Kit (SDK) - more commonly referred to as .NET framework SDK – that provides classes, interfaces and language compilers necessary to program for.NET.

- **Development tools**

If an application that requires speedy delivery to the customers and has features like integration with some version control software then simple Notepad may not serve our purpose. In such cases the Integrated Development Environment (IDE) allows Rapid Action Development (RAD). The new Visual Studio.NET is such an IDE. VS.NET is a powerful and flexible IDE that makes developing .NET applications.
• **Some of the features of VS.NET**
  - Drag and drop design
  - IntelliSense features
  - Syntax highlighting and auto-syntax checking
  - Excellent debugging tools
  - Integration with version control software such as Visual Source Safe (VSS)
  - Easy project management

• **Editions of VS.NET available are:**
  - VS.NET Professional
  - VS.NET Enterprise Developer
  - VS.NET Enterprise Architect

The Visual Studio .NET Professional edition offers a development tool for creating various types of applications mentioned previously. Developers can use Professional edition to build Internet and develop applications quickly and create solutions that span any device and integrate with any platform.

Visual Studio .NET Enterprise Developer (VSED) edition contains all the features of Professional edition plus has additional capabilities for enterprise development. The features include things such as a collaborative team development, third party tool integration for building XML Web services and built-in project templates with architectural guidelines and spanning comprehensive project life-cycle.
The Visual Studio .NET Enterprise Architect (VSEA) edition contains all the features of Visual Studio .NET Enterprise Developer edition and additionally includes capabilities for designing, specifying, and communicating application architecture and functionality. The additional features include Visual designer for XML Web services, Unified Modeling Language (UML) support and enterprise templates for development guidelines and policies.

Special language editions available are:
- Visual Basic.NET Standard Edition
- Visual C# Standard Edition
- Visual C++ .NET Standard

4.2.1 Flavors of .NET

- Rich functionality out of the box

.NET framework provides a rich set of functionality out of the box. It contains hundreds of classes that provide a variety of functionality ready to use in the applications.

- Easy development of Web applications

ASP.NET is a technology available on .NET platform for developing dynamic and data driven Web applications. ASP.NET provides an event driven programming model that simplifies the development of Web pages (now called as Web forms) with complex user interface. ASP.NET server controls provide advanced user interface
elements (like calendar and grids) that save lot of coding from programmer’s side.

- **OOPs Support**

  .NET provides a fully object oriented environment. The philosophy of .NET is – “Object is mother of all.” Languages like Visual Basic.NET now support many of the OO features. Even primitive types like integers and characters can be treated as objects.

- **Multi-Language Support**

  It supports multiple languages. .NET environment. Currently four languages are available right out of the box namely – Visual Basic.NET, C# (pronounced as C-sharp), Jscript.NET and Managed C++ (a dialect of Visual C++). There are many vendors that are working on developing language compilers for other languages (20+ language compilers are already available). The beauty of the multi language support lies in the fact that even though the syntax of each language is different, the basic capabilities of each language remain at par with one another.

- **Multi-Device Support**

  Modern life style is increasingly utilizing mobile and wireless devices such as PDAs, mobiles and handheld PCs. .NET provides promising platform for programming such devices. .NET Compact Framework and Mobile Internet Toolkit are step ahead in this direction.
• **Automatic memory management**

While developing applications developers had to keep an eye on system resources like memory. Memory leaks constitutes the major reasons in failure of applications. .NET takes this worry away from developers by handling memory on its own. The garbage collector takes care of freeing unused objects at appropriate intervals.

• **Compatibility with COM and COM+**

Under .NET, COM components and ActiveX controls can be used. This allows us to use the existing investment in .NET applications. .NET still relies on COM+ for features like transaction management and object pooling. In fact, it provides enhanced declarative support for configuring COM+ application right from the source code.

• **No more DLL Hell**

DLL conflicts are a common fact in COM world. The main factor behind this was the philosophy of COM – “one version of component across machine”. Also, COM components require registration in the system registry. .NET ends this DLL hell by allowing applications to use their own copy of dependent DLLs. Also, .NET components do not require any kind of registration in system registry.

• **Strong XML support**

Nowadays it is hard to find a programmer who is unaware of XML. XML has gained such a strong industry support that almost all the vendors have released some kind of upgrades or patches to their existing
software to make it “XML compatible”. Currently, .NET is the only platform that is built with XML right into the core framework. .NET tries to harness power of XML in every possible way. In addition to providing support for manipulating and transforming XML documents, .NET provides XML Web services that are based on standards like HTTP, XML and SOAP.

- **Ease of deployment and configuration**

  Deploying windows applications especially that used COM components have always been a tedious task. Since .NET does not require any registration as such, much of the deployment is simplified. This makes XCOPY deployment viable. Configuration is another area where .NET – especially ASP.NET – shines over traditional languages. The configuration is done via special files having special XML vocabulary. Since most of the configuration is done via configuration files, there is no need to sit in front of actual machine and configure the application manually. This is more important for Web applications; simply FTPping new configuration file makes necessary changes.

- **Security**

  Microsoft has taken great efforts to make the .NET platform safe and secure for enterprise applications. Features such as type safety, code access security and role-based authentication make overall application more robust and secure.
4.2.2 Microsoft.Net Solutions

• Single Programming Model

A related goal is to have development for the Internet environment look very much like development for other types of software. Likewise, developing user interfaces in Windows Forms is very similar to developing them into Web Forms. There are commonly used controls, such as Labels and Text Boxes, in both, with similar sets of properties and method. The amount of commonality makes it easy to transit between the two types of development, and easier for traditional VB developers to start using Web Forms.

• Distributed Systems

The Vision of Microsoft.NET is globally distributed systems, using XML as the universal glue to allow functions running on different computers across an organization or across the world to come together in a single application. In this vision, systems from servers to Wireless Palmtops, with everything in between, will share the same general platform, with versions of .NET available for all of them, and with each of them able to integrate transparently with the others.

• Richer User Interface

Web Forms are a giant step towards much richer Web-based user interfaces. Their built-in intelligence allows rich, browser-independent screens to be developed quickly, and to be easily integrated with compiled code.
• **Easy deployment**

Executable modules in .NET are self-describing. Once the Common Language Runtime knows where a module resides, it can find out everything else it needs to know to run the module, such as the module’s object interface and security requirements, from the module itself. That means a module can just be copied to a new environment and immediately executed.

• **Support for Multiple Languages**

The CLR executes a binary code called MSIL (Microsoft intermediate language), and that code looks the same regardless of the original source language. All .NET–enabled languages use the same data types and the same interfacing conventions. This makes possible for all .NET language to interoperate transparently. One language can call another easily, and languages can even inherit classes written in another language and extend them current platform anywhere near this level of language interoperability.

• **Extendibility**

The completely object-based approach of .NET is designed to allow base functionality to be extended through inheritance (unlike COM) and the platform’s functionality is appropriately partitioned to allow various parts (such as the just-in-time compilers discussed in the next section) to be replaced as new versions are needed. It is likely that, in the future, new ways of interfacing to the outside world will be added
to the current trio of windows Form, Web Forms, and Web Services such as universal Canvas.

- **Portability of compiled applications**

  .NET allows the future possibility of moving software to other hardware and operating system platforms. The ultimate goal is that compiled code produced on one implementation of .NET could be moved to another implementation of .NET on a different operating system merely by copying the compiled code over and running it.

- **Integrity with COM**

  .NET integrates very well with COM-based software. Any COM component can be treated as a .NET component by other .NET components. The .NET framework wraps COM components and exposes an interface that .NET components can work with. This is absolutely essential to the quick acceptance of .NET, because it makes .NET interoperable with a tremendous amount of older COM-based software.

4.2.3 **Other benefits of using .NET architecture**

- The Microsoft .NET platform's reliance on XML for data exchange—an open standard managed by the WWW Consortium (W3C)—and modular XML Web services removes barriers to data sharing and software integration.
- The .NET platform, through the .NET Framework's common language runtime, enables XML Web services to interoperate
whatever be their source language. Developers can build reusable XML Web services instead of monolithic applications.

- Easier to build sophisticated development tools – debuggers and profilers can target the Common Language Runtime, and thus become accessible to all .NET-enabled languages.
- A Potentially better performance in system level code for memory management, garbage collection, and the like have yielded an architecture that should meet or exceed performance of typical COM-based applications today.
- Fewer bugs, as whole classes of bugs should be unknown in .NET.
- Faster development using development tool like visual studio.net

4.3 INTRODUCTION TO MICROSOFT VISUAL STUDIO

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, Web sites, Web applications, and Web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

The Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools
include a forms designer for building GUI applications, Web designer, class designer, and database schema designer.

It allows plug-ins to be added that enhance the functionality at almost every level - including adding support for source control systems (like Subversion and Visual SourceSafe) to adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports languages by means of language services, which allow any programming language to be supported (to varying degrees) by the code editor and debugger, provided a language-specific service has been authored. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), and C# (via Visual C#). Support for other languages such as Chrome, F#, Python, and Ruby among others has been made available via language services which are to be installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Language-specific versions of Visual Studio also exist which provide more limited language services to the user. These individual packages are called Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

Visual Studio does not support any programming language, solution or tool intrinsically. Instead, it allows various functionalities to be plugged in. Specific functionality is coded as a VSPackage. When installed, the functionality is available as a Service. The IDE provides
three services: SVsSolution, which provides the ability to enumerate projects and solutions; SVsUIShell, which provides windowing and UI functionality (including tabs, toolbars and tool windows); and SVsShell, which deals with registration of VSPackages.

In addition, the IDE is also responsible for coordinating and enabling communication between services. All editors, designers, project types and other tools are implemented as VSPackages. The Visual Studio uses COM to access the VSPackages. The Visual Studio SDK also includes the Managed Package Framework (MPF), which is a set of managed wrappers around the COM-interfaces that allow the Packages to be written in .NET languages. However, MPF does not provide all the functionalities exposed by the Visual Studio COM interfaces. The services can then be consumed for creation of other packages, which add functionalities to the Visual Studio IDE.

Support for programming languages is added by using a specific VSPackage called a Language Service. A language service defines various interfaces which the VSPackage can implement to add support for various functionality. Functionalities that can be added this way includes syntax coloring, statement completion, brace matching, parameter information tooltips, member lists and error markers for background compilation. If the interface is implemented, the functionality will be available for the language.

Language services are to be implemented on a per-language basis. The implementations can reuse code from the parser or the compiler for
the language. Language services can be implemented either in native code or managed code. For native code, either the native COM interfaces can be used, or the Babel Framework (part of Visual Studio SDK) can be used. For the managed code, the MPF includes wrappers for writing managed language services.

The Visual Studio does not include any source control support built in but it defines the MSSCCI (Microsoft Source Code Control Interface) by implementing which source control systems can integrate with the IDE. MSSCCI defines a set of functions that are used to implement various source control functionalities. MSSCCI was first used to integrate Visual SourceSafe with Visual Studio 6.0 but was later opened up via the Visual Studio SDK. Visual Studio .NET 2002 used MSSCCI 1.1, and Visual Studio .NET 2003 used MSSCCI 1.2. Both Visual Studio 2005 and 2008 use MSSCCI Version 1.3, which adds support for rename and delete propagation as well as asynchronous opening.

Visual Studio supports running multiple instances of the environment (each with its own set of VSPackages). The instances use different registry hives to store their configuration states and are differentiated by their AppId (Application ID). The instances are launched by an AppId-specific .exe that selects the AppId, sets the root hive and launches the IDE. VSPackages registered for one AppId are integrated with other VSPackages for that AppId.
The various product editions of Visual Studio are created using the different AppIds. The Visual Studio Express edition products are installed with their own AppIDs, but the Standard, Professional and Team Suite products share the same AppId.

Consequently, the Express editions can be installed side-by-side with other editions, unlike the other editions which update the same installation. The professional edition includes a superset of the VSPackages in the standard edition and the team suite includes a superset of the VSPackages in both other editions. The AppId system is leveraged by the Visual Studio Shell in Visual Studio 2008.

4.3.1 Features of visual studio

- Code editor

The Visual Studio, like any other IDE, includes a code editor that supports syntax highlighting and code completion using IntelliSense not only for variables, functions and methods but also for language constructs like loops and queries. IntelliSense is supported for the included languages, as well as for XML and for Cascading Style Sheets and JavaScript when developing Web sites and Web applications. Auto complete suggestions are popped up in a modeless list box, overlaid on top of the code editor.

From Visual Studio 2008 onwards, it can be made temporarily semi-transparent to see the code obstructed by it. The code editor is used for all supported languages.
The Visual Studio code editor also supports setting bookmarks in code for quick navigation. Other navigational aids include collapsing code blocks and incremental search, in addition to normal text search and regex search. The code editor also includes a multi-item clipboard and a task list. The code editor supports code snippets, which are saved templates for repetitive code and can be inserted into code and customized for the project being worked on. A management tool for code snippets is built in as well.

These tools are surfaced as floating windows which can be set to automatically hide when unused or docked to the side of the screen. The Visual Studio code editor also supports code refactoring including parameter reordering, variable and method renaming, interface extraction and encapsulation of class members inside properties, among others.

Visual Studio features background compilation (also called incremental compilation). As code is being written, Visual Studio compiles it in the background in order to provide feedback about syntax and compilation errors, which are flagged a red wavy underline. Warnings are marked with a green underline. Background compilation does not generate executable code, since it requires a different compiler than the one used to generate executable code.

Background compilation was initially introduced with Microsoft Visual Basic but has now been expanded for all included languages.
• **Debugger**

Visual Studio includes a debugger that works both as a source-level debugger as well as machine-level debugger. It works with both a managed code as well as native code and can be used for debugging applications written in any language supported by Visual Studio. In addition, it can also attach to running processes, monitor and debug those processes. If source code for the running process is available, it displays the code as it is being run. If a source code is not available, it can show the disassembly. The Visual Studio debugger can also create memory dumps as well as load them later for debugging. Multi-threaded programs are also supported. The debugger can be configured to be launched when an application running outside the Visual Studio environment crashes.

• **Data tool tips in Visual Studio**

The debugger allows setting breakpoints (which allow execution to be stopped temporarily at a certain position) and watches (which monitor the values of variables as the execution progresses). Breakpoints can be conditional, that is, they get triggered when the condition is met. Code can be stepped over, i.e., run one line (of source code) at a time. It can either *step into* functions to debug inside it, or *step over* it, i.e., the execution of the function body isn't available for manual inspection. The debugger supports *Edit and Continue*, i.e., it allows code to be edited as it is being debugged.
While debugging, if any variable is hovered over by the mouse pointer, its current value is displayed in a tooltip ("data tooltips"), where it can also be modified if desired. During coding, the Visual Studio debugger lets certain functions be invoked manually from the Immediate tool window. The parameters to the method are supplied at the Immediate window.

- Designer

The Visual Studio includes a host of visual designers to aid in the development of applications.

- WinForms Designer

The WinForms designer is used to build GUI applications using WinForms. It includes a palette of UI widgets and controls (including buttons, progress bars, labels, layout containers and other controls) that can be dragged and dropped on a form surface. The Layout can be controlled by housing the controls inside other containers or locking them to the side of the form. Controls that display data (like textbox, list box, grid view, etc.) can be data bound to data sources like databases or queries. The UI is linked with code using an event-driven programming model. The designer generates either C# or VB.NET code for the application.

- WPF Designer
The WPF designer, codenamed *Cider*, was introduced with Visual Studio 2008. Like the WinForms designer it supports use of the drag and drop metaphor. It is used to author user interfaces targeting Windows Presentation Foundation. It supports all WPF functionalities including data binding and automatic layout management. It generates XAML code for the UI. The generated XAML file is compatible with Microsoft Expression Design, the designer-oriented product. The XAML code is linked with code using a code-behind model.

- **Web designer**

  The Visual Studio also includes a Web site editor and designer that allows Web pages to be authored by dragging and dropping widgets. It is used for developing ASP.NET applications, and supports HTML, CSS and JavaScript. It uses a code-behind model to link with ASP.NET code. Visual Studio 2008 onwards, the layout engine used by the Web designer is shared with Microsoft Expression Web.

- **Class designer**

  The Class Designer is used to author and edit the classes (including its members and their access) using UML modeling. The Class Designer can generate C# and VB.NET code outlines for the classes and methods. It can also generate class diagrams from handwritten classes.

- **Data designer**
The data designer can be used to graphically edit database schemas, including typed tables, primary and foreign keys and constraints. It can also be used to design queries from the graphical view.

- **Mapping designer**

Visual Studio 2008 onwards, the mapping designer is used by LINQ to SQL to design the mapping between database schemas and classes that encapsulate the data.

## 4.4 INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

- **Classes**

One of the major problems in the earlier approach was also the data and the functions working upon the data are separate. This leads to the necessity of checking the type of data before operating upon the data. The first rule of the object-oriented paradigm says that if the data and the functions acting upon the data can go together, let them be together. The unit, which contains the data and its functions together is a *class*. A class can also be defined as a programmatic representation of an entity and the behavior of that entity can be represented by the functions in the class.

- **Encapsulation**

Encapsulation is an implementation of abstraction. It leads directly to two main advantages: Data Hiding and Data Security.
• **Inheritance**

In real life, inheritance allows us to reuse things that belong to a particular entity. Also, in object oriented world, a class can inherit the properties and functionalities defined in some another class so that they can be reused. Reusability cannot be done unless the classes are of the same type. So the class which would be reusing the functionalities of the other classes are derived class. The class that is being “derived from” is termed as the base class.

Inheritance directly results in the following benefits: --

• Reusability:
• Enhancement and Specification:
• Avoiding type inspection
• Polymorphism

The word “polymorphism” means “different forms”. Applied in object-oriented paradigm, it means the ability of an entity to exhibit different forms at runtime.

One major reason to have this is to eliminate the type inspection.

### 4.4.1 Introduction to unified modeling language

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems. One of the purposes of UML was to provide the development community with a stable and common design language that could be used to develop and build computer applications.
UML brought forth a unified standard modeling notation that IT professionals had been wanting for years. UML was meant to be a unifying language enabling IT professionals to model computer applications. One reason UML has become a standard modeling language is that it is programming-language independent. The Unified Modeling Language (UML) is a Graphical language for visualizing, specifying and constructing the artifacts of a software-intensive system.

The Unified Modeling Language offers a standard way to write a system's blueprints, including conceptual components such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components. UML combines the best practice from data modeling concepts such as entity relationship diagrams, business modeling (work flow), object modeling and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.

4.4.2 UML diagrams - Overview

UML has 13 types of diagrams divided into three categories: Six diagram types represent the structure application, seven represent general types of behavior, including four represent different aspects of interactions.

a. Structure diagrams
Structure diagrams emphasize what things must be in the system being modeled:

- Class diagram: describes the structure of a system by showing its system's classes, their attributes, and the relationships among the classes. The class diagram shows how the different entities (people, things, and data) are related to one another; in other words, it shows the static structures of the system. A class diagram can be used to display logical classes. Class diagrams can also be used to show implementation classes, which are the things that programmers typically deal with. An implementation class diagram will probably show some of the same classes as the logical classes diagram. The implementation class diagram won't be drawn with the same attributes, however, because it will most likely have references to things like Vectors and HashMaps.

- Component diagram: depicts how a software system is split up into components and shows the dependencies among these components. A component diagram provides a physical view of the system. The diagram can be shown at a very high level, with just the large grain components, or it can be shown at the component package level.

- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.

- Deployment diagram: serves to model the hardware used in system implementations, the components deployed on the hardware, and the associations among those components. The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate.
with each other. Since the diagram models the physical runtime, a system's production staff will make considerable use of this diagram.

- **Object diagram:** shows a complete or partial view of the structure of a modeled system at a specific time.
- **Package diagram:** depicts how a system is split up into logical groupings by showing the dependencies among these groupings.

### b. Behavior diagrams

Behavior diagrams emphasize what must happen in the system being modeled:

- **Activity diagram:** represents the business and operational step by step workflows of components in a system. An activity diagram shows the overall flow of control. Activity diagrams show the procedural flow of control between two or more class objects while processing an activity. Activity diagrams can be used to model higher-level business processes at the business unit level, or to model low-level internal class actions.
- **State diagram:** models the different states that a class can be in and how that class transitions from state to state. Only classes with "interesting" states -- that is, classes with three or more potential states during system activity -- should be modeled.
- **Use case diagram:** shows the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases. A use case illustrates a unit of functionality provided by the system.
The main purpose of the use case diagram is to help development teams visualize the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use case diagrams generally show groups of use cases, either all use cases for the complete system, or a breakout of a particular group of use cases with related functionality.

**c. Interaction diagrams**

- Interaction diagrams: is a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled:

- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.

- Interaction overview diagram: is a type of activity diagram in which the nodes represent interaction diagrams.

- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the life spans of objects relative to those messages. Sequence diagrams show a detailed flow for a specific use case or even just part of a specific use case. They are almost self explanatory; they show the calls between the different objects in their sequence and can show, at a detailed level, different calls to different objects. A sequence diagram has two dimensions: The vertical dimension shows the sequence of
messages/calls in the time order that they occur; the horizontal
dimension shows the object instances to which the messages are sent.

- Timing diagrams: are a specific type of interaction diagram, where
  the focus is on timing constraints.