Palestine Polytechnic University
College of Engineering & Technology
Electrical & Computer Engineering Department

Graduation Project

Static Class Diagrams Constructor CASE Tool (SCDC)

Project Team:
Ibrahim M. Saraheen
Sari M. Jabareen
Sari M. Adam

Project Supervisor
Eng. Khalid Daghameen

Hebron – Palestine

2009

Static Class Diagrams Constructor CASE Tool (SCDC)

Project Team:
Ibrahim Mohammad Saraheen
Sari Mahmoud A'dam
Sari Majed Jabareen

Supervisor:
Eng. Khalid Daghameen

Graduation Project Report

Submitted to the Department of Electrical & Computer Engineering in the College of Engineering and Technology at Palestine Polytechnic University

Hebron – Palestine

June, 2009

جامعة بوليتكنك فلسطين
الخليل – فلسطين
كلية الهندسة و التكنولوجيا
دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع:

Static Class Diagrams Constructor CASE Tool

أسماء الطلبة:
إبراهيم محمد السراحين ساري ماجد جبارين ساري محمود العدم

اسم المشرف : م. خالد الدغامين

بناء على نظام كلية الهندسة و التكنولوجيا و إشراف و متابعة المشرف المباشر على المشروع وموافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب وذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب.

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Dedication

To Our Families and Friends

ACKNOWLEDGMENT

Our appreciation to:

Palestine Polytechnic University

College of Engineering & Technology

Electrical & Computer Engineering Department

Our supervisor: Eng. Khalid Daghameen for his great help and supports

ABSTRACT

There is a great necessity for using object-oriented methods in developing software systems. The most important component which must be developed is the static class diagram. The development of such class diagrams in a systematic way is very crucial in an object-oriented development methodology. This project aims of implementing a new software approach to obtain a static class diagram ,their relation and their skeleton programming codes.

This project is based mainly on a scientific paper titled : " A Systematic Approach for Constructing Static Class Diagrams from Software Requirements " by : Dr. Nabil Arman and Eng. Khalid Daghameen .

الملخص

هناك حاجة ملحة لاستخدام أساليب برمجة الكيانات في تطوير الأنظمة البرمجية. و من الأمور المهمة التي يجب تطويرها الكيانات الثابتة في برمجة الكيانات و تطوير هذه الكيانات بطريقة ممنهجة و نظامية عنصر هام وحاسم في تطوير أساليب ومناهج برمجة الكيانات. مشروعنا يهدف إلى تطبيق طريقة برمجية جديدة للحصول على الكيانات الثابتة وعلاقاتها و الأكواد البرمجية التابعة لها.

هذا المشروع يعتمد بشكل رئيسي على ورقة علمية بعنوان "طريقة ممنهجة لبناء الكيانات الثابتة من متطلبات البرمجة " قدمت من قبل د. نبيل عرمان و م. خالد الدغامين .

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1. **Overview**
1.2. **Project Importance**
1.3. **Project Goals**
1.4. **Literature Review**
1.5. **Time plan**
1.6. **Estimated Cost**
1.7. **Project Risk Management**
1.8. **Road Map**

Chapter One

Introduction

1.1. Overview

The object-oriented methods have been necessary for software systems development , since these methods are implemented in programming languages like C++ ,VB.Net and JAVA etc … . Also it is implemented in the web development and nowadays it is even in database management systems and other software systems . This makes a need for development of object-oriented methods and approaches in object-oriented software systems.

Many object-oriented development methodologies have been developed in the past two decades to develop object-oriented systems, and to support object-oriented programming. A system development methodology (SDM) has been defined as "…a systematic approach to conducting at least one complete phase (e.g., requirements analysis, design) of system development, consisting of a set of guidelines, activities, techniques and tools, based on a particular philosophy of system development and the target system" [8]. From this definition, it is generally recognized that there is a great emphasis on "being systematic" for any stage of system development methodology[1].

Static class diagrams represent an essential component in an object-oriented system design. The development of such class diagrams in a systematic way is very crucial in an object-oriented development methodology [1]. In this project , we are to implement a new approach for obtaining these static class diagrams in a systematic way which is very essential in object-oriented development practice .

1.2. Project Importance

Obtaining static class diagrams is needed in object-oriented software systems development and some software tools try to ease this –obtaining static class diagrams- but it still depends mainly on software engineers/practitioners, whereas this project not only implements new approach which is less dependent on human intervention but also it is implemented in a systematic way.

This project will help software engineers/practitioners and programmers to analyze and design their object-oriented software and also to reduce the cost and the waste of time for doing these processes by humans . The idea of this systematic approach for constructing static class diagrams from software requirements is a novel one and it was never implemented before .

1.3. Project Goals

- To survey the object-oriented development methodologies and find their strengths and weaknesses against our approach .
- To provide a tool that implements this new approach for obtaining the static class diagrams from text/problem statement in the software requirements .
- Making this tool is able to generate skeleton programming codes .
- Making the process of building object-oriented software easier and simpler.

1.4. Literature Review

In this section we will preview some of tools that related to the project main idea which is obtaining class diagrams .

1.4.1. *"Rational Rose"* [3]

Rational Rose is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprise-level software applications. In much the same way a theatrical director blocks out a play, a software designer uses Rational Rose to visually create (model) the framework for an application by blocking out classes with actors (stick figures), use case elements (ovals), objects (rectangles) and messages/relationships (arrows) in a sequence diagram using drag-and-drop symbols. Rational Rose documents the diagram as it is being constructed and then generates skeleton code in the designer's choice of C++, Visual Basic, Java, Oracle8, CORBA or Data Definition Language.

Two popular features of Rational Rose are its ability to provide iterative development and round-trip engineering. Rational Rose allows designers to take advantage of iterative development (sometimes called evolutionary development) because the new application can be created in stages with the output of one iteration becoming the input to the next. (This is in contrast to waterfall development where the whole project is completed from start to finish before a user gets to try it out.) Then as the developer begins to understand how the components interact and makes modifications in the design, Rational Rose can perform what is called "round-trip engineering" by going back and updating the rest of the model to ensure the code remains consistent.

Rational Rose is extensible, with downloadable add-ins and third-party partner applications. It supports COM/DCOM (ActiveX), JavaBeans, and Corba component standards.

1.4.2. *"Rational Unified Process (RUP)"* [4]

Rational Unified Process (RUP) is an object-oriented and Web-enabled program development methodology. According to Rational (developers of Rational Rose and the Unified Modeling Language), RUP is like an online mentor that provides guidelines, templates, and examples for all aspects and stages of program development. RUP and similar products -- such as Object-Oriented Software Process (OOSP), and the OPEN Process -- are comprehensive software engineering tools that combine the procedural aspects of development (such as defined stages, techniques, and practices) with other components of development (such as documents, models, manuals, code, and so on) within a unifying framework.

RUP establishes four phases of development, each of which is organized into a number of separate iterations that must satisfy defined criteria before the next phase is undertaken: in the inception phase, developers define the scope of the project and its business case; in the elaboration phase, developers analyze the project's needs in greater detail and define its architectural foundation; in the construction phase, developers create the application design and source code; and in the transition phase, developers deliver the system to users. RUP provides a prototype at the completion of each iteration. The product also includes process support for Java 2 Enterprise Edition (J2EE) and BEA (Web Logic) development, and supplies an HTML-based description of the unified process that an organization can customize for its own use.

1.5. Time Plan

In this section we will preview the system schedule for the developers, take in Care the time period given for delivery the system, and the dependences between Tasks.

Table 1.1. Tasks description (first semester)

Task ID	Task description
T1.1	Selecting the project.
T1.2	Collecting information, literature review and related theory.
T1.3	System analysis
T1.4	System design
T1.5	Writing documentation

Table 1.2. Time plan (first semester)

Task/week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T1.1																
T1.2																
T1.3																
T1.4																
T1.5																

Table 1.3. Tasks description (second semester)

Task ID	Task description
T2.1	System design and analysis(cont.).
T2.2	Component implementation.
T2.3	Components testing.
T2.4	Integrity testing.
T2.5	System review and incorporating feedback.
T2.6	Requirements refinement.
T2.7	Writing documentation

Table 1.4. Time plan (second semester)

Task/week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T2.1																
T2.2																
T2.3																
T2.4																
T2.5																
T2.6																
T2.7																

1.6. Estimated Cost

In this section we will preview the system cost estimation in term of system Hardware , system software , and human resources.

1.6.1. Software resources

The following table contains the software that will be used in our system and its Costs.

Table 1.5. Software resources cost

Software component	Cost
Microsoft Windows XP Professional	Available from the university
Microsoft Office 2007 enterprise	Available from the university
Microsoft Office Visio 2007	Available from the university
Microsoft Visual Studio 2005	Available from the university
Total	$0

1.6.2. Hardware resources

The following table contains the main requirements for our system and their Costs.

Table 1.6. Hardware resources cost

Hardware component	Cost
1 PC with core 2 duo CPU 2.6 GHz, 160 GB HDD, 2 GB RAM, monitor, keyboard and mouse.	Available from the university
Printing cost	$40
Total	**$40**

1.6.3 Human resources

The project team consists of three students, work 32 weeks for two semesters, 3 hours a day with $1 per hour, assuming 4 working days a week .

each student get : (32weeks * 4days * 3hours * $1) = $384 .

and the total for 3 students cost is: $384 * 3 = **$1152** .

1.6.4. Other resources

The following table contains other resources using in our system and their Costs.

Table 1.7. Other resources cost

Resource	Cost
Transportation	$100
Internet usage	Available from the university
Total	**$100**

1.6.5. Total Costs

Table 1.8. Total cost

Resource	Cost
Software resources	$0
Hardware resources	$40
Human resources	$1152
Other resources	$100
Total	**$1292**

1.7. Project Risk Management

The project may face some problems and risks that we have to declare in the early time of the project designing and manipulation, and we must avoid those problems so that the system could work in its high efficiency, so when we find a risk we will try to solve it without affecting the total project as much as we can.

Table 1.9. Risks analysis

Risk ID	Risk description	Probability	Impact ID
R1	One of the group members dies or gets sick.	40%	1/2
R2	Lack of experience in using programming tools.	60%	3
R3	Hardware failure.	5%	4
R4	Needed time for completing the project is longer than expected.	30%	2
R5	Results are not correct.	30%	2
R6	Closure of the university.	40%	4

Table 1.10. Impacts ID's

Impact ID	Impact
1	Catastrophic
2	Critical
3	Marginal
4	Negligible

Table 1.11. Risks effects minimizations strategies

Risk ID	Strategy to minimize effect on the project
R1	the team must divide the work so that if one was absent the other do his work.
R2	Read and study more on using these programming tools.
R3	There should be a backup plan.
R4	Main features will be delivered and other features will be delayed.
R5	The project should be tested at every stage of development.
R6	The team can meet at any other place with all required resources.

1.8. Road Map

The project consists of six chapters, each chapter talks about a specific area of the project.

Chapter One: "Introduction", this chapter gives an introduction about the system, its importance, other related systems, project time plan and project cost.

Chapter Two: "Theoretical Background", this chapter gives a clear picture about the system theoretical background related to the main concepts of object-oriented and class diagrams .

Chapter Three: "Design Concepts", this chapter shows system block diagram, design options that can be used in the system, and how each system components interface with each other.

Chapter Four: "Detailed system design", this chapter describes the program that operates the project, GUI and flowcharts that describe project systems processes..

Chapter five: "System Implementation and Testing", this chapter discusses the actual project testing and implementation in details.

Chapter Sex: "Conclusion and Future Work", this chapter is simplifying the conclusions and results achieved after implementing the entire project and gives suggestions for the future system developing.

Chapter 2

Theoretical Background

2.1. **Structured Programming**

2.2. **Object Oriented Programming (OOP)**

2.3. **UML Class Diagram**

Chapter Two

Theoretical Background

This chapter provides an illustrative theoretical background for the project related topics.

2.1. Structured Programming

The fundamental principle of structured programming is that at all times and under all circumstances, the programmer must keep the program within his intellectual grasp. The well-known methods for achieving this can be briefly summarized as follows: 1) top-down design and construction, 2) limited control structures, and 3) limited scope of data structures.

The normal or procedural method of programming with other or even with object oriented programming (OOP) supported language is conceptually known as structured programming. Though there are some different options regarding 'structure' or data structure and procedure, the whole theme of such concept is actually almost the same. The main program itself is a function or procedure in structured programming. A typical example of structured programming is FORTRAN.

Around 1970, computer science experts started discussing the new idea of structured programming. After some debate, the consensus emerged, at least within the field of computer science, that structure is a good practice to encourage. Pascal was one of the first widely popular languages to adopt an aggressive restructured approach. Keeping a historical perspective, you must remember that at the time that

the academic community equated unstructured with bad programming style, probably 99 percent of the world's programmers learned how to program in a college setting. If you learned how to write software, you learned structure. Everybody agreed that it was the way to program.

In the late 1970s and early 1980s, something happened. Low-cost microcomputers from Commodore, Apple, Radio Shack, and other companies made home computing affordable. Most personal computers came with BASIC-the language invented before structured programming. Most computer owners learned BASIC from books, magazines, or friends-outside of an academic environment. Suddenly there were millions of people happily writing unstructured programs. Computer scientists were aghast.

Let's propose two general rules of structured programming, both of which relate to coding style. The first rule is that when you program, you should be organized and separate the program into bite-sized sections. The second rule is that you should write programs that are easy for other people to read.

The first is the most important rule. It's sometimes called modular programming. As you write a program, you split it up into modules or subroutines. Instead of running straight through from line 10 to line 5000, divide it up. Replace the mammoth 500-line program with five 100-line routines, each of which in turn has about five 20-line routines .If others, including teachers or programmers, will see your program, they'll be able to read it more easily if you write in a structured style. Most, it's embarrassing to have someone look at a sloppily-written program you wrote. What about programs you write for your own use that no one else will see? The time may come when you need to change something. A spaghetti program you wrote six months before is nearly impossible to follow.

The proponents of unstructured style say, "If the program works, it works. The rules of structure are a fetter on my creativity. Besides, if no one else ever sees the program, it doesn't matter if the listing looks pretty" . The conceptual meaning of normal programming language and structured programming language is almost the same. Using of, so called, procedures, functions or structures are not the meaning but the usual concept and the definition of structured programming such as- Pascal, Fortran, C and in some case Basic or Lispo also.

The definition and the difference of structured programming languages and the object oriented ones will get clearer as we will further proceed in depth of our project.

2.2. Object Oriented Programming (OOP)

2.2.1. OOP Overview

Object-orientation or object oriented programming (OOP) is introduced as a new programming concept which should help one in developing high quality software. Object-orientation is also introduced as a concept which makes developing of projects easier. Object oriented programming attempts to solve the problems with only one approach; dividing the problems in sub-modules and using different objects. Objects of the program interact by sending messages to each other.

Object oriented programming (OOP), first developed in the 1960s, and reorganizes the programming problem to allow for a higher level of abstraction. Programming with objects is quite like working with real-world objects. It groups operations and data into modular units called objects. These objects can be combined into structured networks to form a complete program, similar to how the pieces in a puzzle fit together to create a picture.

By breaking down complex software projects into small, self-contained, and modular units, object orientation ensures that changes to one part of a software project will not adversely affect other portions of the software. Object orientation also aids software reuse. Once functionality is created in one program, it can easily be reused in other programs.

In contrast to procedural programming focus on the interaction between data and functions, the design of objects and the interactions between those objects become the primary elements of object oriented program design. Object Oriented programming groups data into classes and operations on those classes. Conceptually, the "same" operation might do different things, depending on the classes of the data it is being applied to. Typically, classes can be defined in terms of other classes (inheritance), so that the data within a particular instance of a class, and the operations applicable to those instances, can be shared between the new class and the classes from which it inherits.

The major motivation factor in the invention of object oriented approach is to salvage some of the flaws encountered in the procedural approach. OOP tracts data as a critical element in the program development and does not allow it to flow freely around the sysytem. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allowes us to decompose a problem into a number of entities called objects and then builds data and functions around these entities.

A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. One of the principal advantages of *object-oriented programming techniques over procedural programming techniques is that they enable programmers to create*

modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

2.2.2. Basic Concepts in OOP

The basic terms that you ought to know about OOP concept are :

- Objects
- Classes
- Inheritance
- Polymorphism

Besides, you need to learn about data abstraction, Data encapsulation, Dynamic binding, Message passing, E-R relationship as they will help you to understand OOP's advantages and OOP concept. All these are detailed in the following sections.:

2.2.2.1. Objects

Objects are the central idea behind OOP. The idea is quite simple. An object is a bundle of variables and related methods. A method is similar to a procedure. A method is an operation which can modify an objects behavior. In other words, it is something that will change an object by manipulating its variables. Like 'functions' in C or C++.

The basic idea behind an object is that of simulation. Most programs are written with very little reference to the real world objects the program is designed to work with; in object oriented methodology, a program should be written to simulate the states and activities of real world objects. This means that apart from looking at data structures when modeling an object, we must also look at methods associated

with that object, in other words, functions that modify the objects attributes. That is, both attributes and functions of an entity (single object) are all together called 'object'.

Now the question arises what are the attributes. Object can be a person, a place, a bank account, a table of data or any item that the program must handle. When a program is executed, the objects interact by sending message to one another. These messages depend on the kind of variables or data types that are being used in that object. The following example will make it clearer. In a program it is an object known as student. Under this object the data or attributes are Name, Date of birth, Marks etc. These attribute i.e. variables will be used to interact with other objects or to manipulate in other functions. In a sense these are the characteristics of the object. Under 'student' the data functions are total, average and display. Both this attributes and functions make up this whole as an object. Figure 2.1. illustrates this .

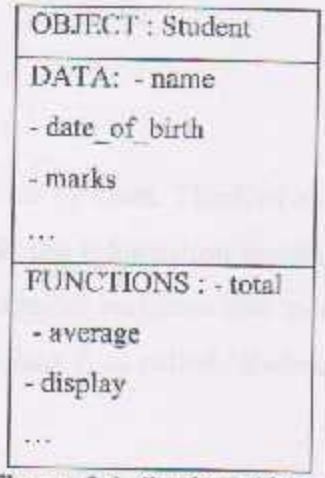

Figure 2.1. Student Object

In other words, an object is an instance of a class. It can be uniquely identified by its name and it defines a state which is represented by the values of its attributes at a particular time.

2.2.2.2. Classes

A class is a blueprint for an object. What this basically means is that we provide a blueprint, or an outline of an object. This blueprint is valid whether we have one or one thousand such objects. A class does not represent an object; it represents all the information a typical object should have as well as all the methods it should have. A class can be considered to be an extremely extended TYPE declaration, since not only are variables held but methods too. It therefore provides implementation details for the data structure used and operations.

A class is an actual representation of an Abstract Data Type (ADT). A class is the implementation of an abstract data type (ADT). It defines attributes and methods which implement the data structure and operations of the ADT, respectively. Instances of classes are called objects. Consequently, classes define properties and behavior of sets of objects.

An object may be defined by class. Think of a class as a blueprint for making object instances. It provides all the information needed to build new instances of an object. Each class defines the internal variables that hold the data of an object instance and the ways, or methods. A class A is called 'abstract class' if it is only used as a super class for other classes.

2.2.2.3. Inheritance

Inheritance is the mechanism which allows a class A to inherit properties of a class B. We say A inherits from B. Objects of class A thus have accesses to attributes and methods of class B without the need to redefine them. The following definition defines two terms with which we are able to refer to participating classes when they use inheritance.

If class A inherits from class B, then B is called super class of A. A is called subclass of B. Objects of a subclass can be used where objects of the corresponding super class are expected. This is due to the fact that objects of the subclass share the same behavior as objects of the super class. In the literature you may also find other terms for super class and subclass. Super classes are also called parent classes. Subclasses may also be called child classes or just derived classes. Subclasses can also override inherited methods and provide specialized implementations for those methods.

2.2.2.4. Polymorphism

Polymorphism means that different objects respond distinctively to the same message. For example, when you send the same message, 'cost' to a spike-bicycle object, mono-cycle object and tandem bicycle object, each one responds appropriately. All of these cycles of the class bicycle have its own individual price.

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operation may be accessed in the same manner even though specific actions associated with each operation may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

2.2.3. More OOP Concepts

2.2.3.1. Data abstraction

The public interface, formed by the collections of messages understood by an object, completely defines how to use this object. Programs that want to manipulate an object, only have to be concerned about which messages this object understands, and do not have to worry about how these tasks are achieved nor the internal structure of the object. The hiding of internal details makes an object abstract, and the technique is normally known as data abstraction. With abstraction you create a well-defined entity which can be properly handled. These entities define the data structure of a set of items. An entity with the properties just described is called an abstract data type (ADT). However, data abstraction and data hiding are almost same. Abstraction refers to the act of representing essential without including the background details or explanations[5].

2.2.3.2. Encapsulation

Encapsulation means that the data and instructions for variable are wrapped up together and treated as a unit. The blueprints for these variables are called objects. Look at the Figure 2.1., The object 'Student' is using three types of data or variable. They are name, date-of-birth and marks. All these variables are used to interact with other objects and are used in respective objective's function. All these data or variable together called encapsulated and the process is known as encapsulation.

2.2.3.3. Messages / Message passing

Objects are autonomous entities which only provide a well-defined interface. We'd like to talk of objects as if they are active entities. For example, objects "are responsible" for themselves, "they" might deny invocation of a method, etc. This distinguishes an object from a module, which is passive. Therefore, we don't speak of procedure calls. We speak of messages with which we 'ask' an object to invoke one of its methods. Software objects interact and communicate with each other using messages. Example- the key word in a programming language 'employee salary

(name)' will first refer first the object name which is here employee, then the message or variable which is here salary and then the information which is here name.

2.2.3.4. Overloading

Overloading, in OOP, means the ability to assign multiple meanings to the names of operators and functions. For example, the operator +, normally associated with arithmetic addition can be overloaded to add one list to another in a certain context.

2.2.3.5. Dynamic Binding

Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time. Think os an object which has a certain function and that was inherited from it's respective class. Thus every object of that class will inherit the function but wont have any effect unless or until it is called. One important reason for having dynamic binding is that it provides a mechanism for selecting between alternatives which is arguably more robust than explicit selection by conditionals or pattern matching. [5]

2.2.4. Advantages of OOP

we now describe the major advantages of OOP. Generally, the advantages are:

- simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear;
- modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system;

- modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
- extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
- maintainability: objects can be maintained separately, making locating and fixing problems easier.
- re-usability: objects can be reused in different programs.
- Provides a clear modular structure for programs. This makes it good for defining abstract datatypes, where implementation details are hidden and the unit has a clearly defined interface.
- Provides a good framework for code libraries, where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.
- Makes it easy to maintain and modify existing code, as new objects can be created with small differences to existing ones.
- The typical programmer's advantages are :
  - Multiple-inheritance, where classes can inherit from multiple super classes.
  - Multi-method dispatch, where operations can be specialized not just on a single object, but on any of the arguments for the operation, or any combination of those arguments.
  - Method combination, in which the means by which several applicable operations to a particular set of arguments can be combined, is itself an object-oriented specification, definable by the programmer.
  - Operations that can specialize on particular instances of classes, rather than on just the broad class of the arguments.
  - shared class variables (slots, attributes) that are shared by all instances of a class, in addition to instance variables which are unique to each instance.
  - Specification of the creation operation for instances of a class as an object-oriented program, definable by the programmer.

- The behaviour of operations on objects (methods) are themselves defined as first-class objects, which can be specialized by the programmer.
- Access to classes themselves as first-class objects. The objects which define the behaviour of classes are themselves classes, usable directly by the programmer.

2.2.5. Drawbacks of OOP

- Much of it is a philosophy. But, the OOP philosophy is not always the best philosophy for many types of applications.
- There are some studies that show that OOP only pays off if a programming project is "well-managed". There are also a few studies that show that even non-OOP COBOL projects can achieve the same famous "code reuse" and "flexibility" goals that OOP lays claim to if managed properly. Thus, OOP's goals may be achievable simply through commitment to certain goals, and OO languages are not a prerequisite. However, let's continue assuming OOP is unique.
- Instance (or shared class) variables of an object can be accessed by class-specialized functions, and called the same way as any other function.
- For simple or easy (even for semi hard) programs, procedural approach can be a good solution. But for the hard ones it is not.
- Some argue that OOP is still important even if not dealing directly with GUI's. In my opinion, much of the type about OOP is faddish OOP in itself does NOT allow programs to do things that they could not do before. OOP is more of a program organizational philosophy rather than a set of new external solutions or operations.
- However, there is a slight cost in terms of efficiency. As objects are normally referenced by pointers, with memory allocated dynamically, there is a small space overhead to store the pointers, and a small speed overhead to find space on the heap (at run-time) to store the objects. For dynamic methods there is an additional small time penalty, as the method to choose is found at run time, by

searching through the object hierarchy (using the class precedence list for the object). These days, these efficiency penalties seem to be of relatively little importance compared with the software engineering benefits.

- OOP is not a new solution for unsolved problems, but only a theory (active theory of course) acting to have better solution for tough programs.

2.2.6. Object-Oriented Analysis and Design (OOAD)

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterized by its class, its state (data elements), and its behavior. Various models can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on what the system does, OOD on how the system does it.

2.2.6.1. Object-Modeling Technique (OMT)

The object-modeling technique (OMT) is an object modeling language for software modeling and designing. It was developed circa 1991 by Rumbaugh, Blaha, Premerlani, Eddy and Lorensen as a method to develop object-oriented systems, and to support object-oriented programming [7].

OMT is a predecessor of the Unified Modeling Language (UML). Many OMT modeling elements are common to UML.

2.2.6.2. Object-Oriented Software Engineering (OOSE)

Ivar Jacobson's Object-Oriented Software Engineering (OOSE) is one of the precursors to the more modern Unified Modeling Language (UML). OOSE includes a requirements, an analysis, a design, an implementation, and a testing model. The Jacobson requirements model includes a problem domain object diagram and use case diagrams. This model defines the limits and functionality of a system. The problem domain object diagram provides a logical view of the system, which is used to specify the use cases for use case diagrams[6].

2.3. UML Class Diagram

The Unified Modeling Language (UML) is a language-agnostic, non-proprietary modeling language that can be used to design object-oriented systems. It is generally regarded as the complete specification of OO, as an abstract design expressed in UML can ideally be implemented in any OO programming languages. UML specifies, among other things, a set of component types and relationships.

In the Unified Modeling Language (UML), a class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes [2].

The UML provide mechanisms to represent class members, such as attributes and methods, and additional information about them. To specify the visibility of a class member (attributes and methods) there are the following notations that must be placed before the member's name :

public

protected

private

package

The UML specify two types of scope for members: instance and classifier. In the case of instance members, the scope is a specific instance. For attributes, it means that its value can vary between instances. For methods, it means that its invocation affects the instance state, in other words, affects the instance attributes. Otherwise, in the classifier member, the scope is the class. For attributes, it means that its value is equal for all instances. For methods, it means that its invocation do not affect the instance state. Classifier members are commonly recognized as "static" in many programming languages. To indicate that a member has the classifier scope, its name must be underlined. Otherwise, as default, the instance scope is considered.

The Generalization relationship indicates that one of the two related classes (the subtype) is considered to be a specialized form of the other (the supertype) and supertype is considered as Generalization of subtype. In practice, this means that any instance of the subtype is also an instance of the supertype. An exemplary tree of generalizations of this form is found in binomial nomenclature: human beings are a subtype of simian, which are a subtype of mammal, and so on. The relationship is most easily understood by the phrase 'A is a B' (a human is a mammal, a mammal is an animal).

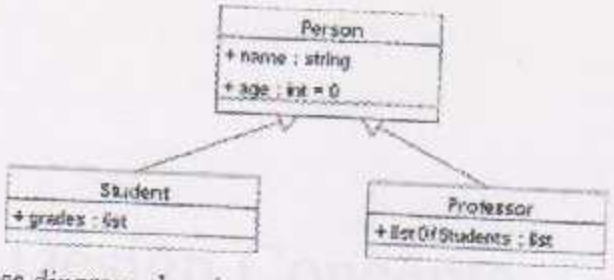

Figure 2.2. Class diagram showing generalization between one superclass and two subclasses [2] .

The UML graphical representation of a Generalization is a hollow triangle shape on the supertype end of the line (or tree of lines) that connects it to one or more subtypes as in figure 2.2. which shows a generalization example between one superclass and two subclasses.

The generalization relationship is also known as the inheritance or "is a" relationship. The super type in the generalization relationship is also known as the "parent", superclass, base class, or base type. The subtype in the generalization relationship is also known as the "child", subclass, derived class, derived type, inheriting class, or inheriting type [3].

Chapter 3

Design Concepts

3.1. **Project Objectives**

3.2. **Architectural Design**

3.3. **System Modeling**

Chapter Three

Design Concepts

3.1. Project Objectives

- Survey the object-oriented development methodologies and find their strengths and weaknesses against our approach .
- Provide a software tool that implements this new approach for obtaining the static class diagrams from text of the problem in the software requirements.
- Making this tool is able to generate skeleton programming codes in some of object oriented programming languages .
- Making the process of building object-oriented software easier and simpler.

3.2. Architectural Design

3.2.1 System's Definition

When the text/problem statement is defined in a software requirements document by the software engineering or the developer … etc A static class diagram is constructed from software requirements and problem statements by following the steps below:[1]

1. Identify the list of all nouns and noun phrases from the problem statement. These nouns and noun phrases represent classes and adjectives represent attributes. A software engineer/practitioner obtains such a list from the

problem statement, use cases, actor-goal list or application narrative description .

2. Make a refinement for the nouns determined in step 1. Such nouns represent values for attributes, or nouns that are identical to other nouns. This step is important to ease the process of having the static diagram, but it is not necessary; nouns that are not related to the problem statement are isolated classes in the static class diagram. This means that they are not part of the system. Nouns that are identical according to the problem statement appear in the static class diagram with the same attributes and associations as other nouns. From that one also needs to pick one of such classes. For example, in a problem statement concerning a university, a teacher may be called by a teacher, or a professor or an instructor. Such nouns mean the same thing in this problem statement .

3. Arrange the nouns from step 2 into a matrix keeping the order of such nouns the same in the rows and columns. The nouns that appear in the left column is called "left cell" in the paper, and the ones that appear in the top row is called "top row". In this early step, one keeps the nouns and noun phrases at the top of the list and the adjectives at the bottom of the list.

4. Fill up the table with the letters (I, H, P, U or keep it empty); by answering a YES/NO questions. The four questions are: Is a "left cell " a "top row"?, Has a "left cell" a " top row" ?, Is a "left cell" part-of a "top row"? and Does a "left cell" use a "top row"?, consecutively. It is important to notice that once a cell is occupied by a letter, you don't need to do the rest of the questions for that particular cell.

5. The answers of the four questions must be according to the problem statement, use cases, actor-goals or application narrative description. In a university problem, the answer for a question such as "Is the graduate-student a student?" is yes. The answer for "Has Student information a date?" is yes. On the other hand, an answer for the question such as "Is a student a graduate-student?" is

no, since there is a student who is not a graduate student for that problem domain.

6. Identify those nouns which are classes and those which are attributes. Any row that has an "H" letter is a class, as for those rows which don't have an "H" letter is left to the software practitioner to identify. Rows that have more than one "I" is a class, also the row which has a single "I" is also a class. As for the adjectives, all of them are attributes .

7. Rearrange the nouns again in the table taking into account that all candidate classes must be sorted according to the number of "H" letters in each row and the others are kept to the end of the list for the classes, while the attributes and methods should occupy the tail of the list. Fill up the lower half of the matrix with letters by asking the simple four questions again.

8. Each identified class in the table should have a class diagram. If you are using UML models, a stand alone class is drawn by the name of the class .

9. Identify the hierarchy by connecting classes using the cells containing "I" letters. An L-shape is formed between two candidate classes. For the hierarchy stair-shape like is obtained by forming more than one L-shape from those "I" cells.

10. An UML static class diagram is obtained from the information you obtained from the previous step.

11. For the "P" cells, arrange them into groups according the groups of "I" cells. For example, a stand alone "P" in a row means that this is an attribute "left cell" is an attribute in the class "top row". For two "P" cells in a row, find the *equivalent of "I" cells that form the lower part of* L shape from a row in the top. Those are part of the lower L-shape, and this attribute is part of the class which is in the higher L-shape of "I" cells. A group of three "P" cells is the same and you need to find the Upper L-shaped class upon identifying the Attribute position of an attribute.

12. For those "P" cells, which are classes, or those classes that have "has-a" associations with another class. The first step is to locate the position of the class which should be connected to another class. This is done using the same as step 11.

3.2.1.1. A Case Study [1]

Assume that the following text/problem statement is taken from a software requirements document:

Vessels are one kind of containers that contain liquids, vessels are different in shapes and sizes. Each vessel has a capacity which is the maximum quantity of the substance that is contained using a vessel. The amount of substance in a vessels wobbles between zero to the maximum of the capacity of a vessel. When someone fills up a vessel, he/she is going to add some substance to the existing substance in the vessel.

A lot of shapes of vessels are available, but we are concerned with few of them, some of them which are rectangular tanks, this type has a rectangular base and a height, the rectangular base is identified by its length and width. A cubic tank which is the same as a rectangular one except that the base is a square base and both length and width have the same value. A cylindrical tank has a circular base and height. A circular base is identified by its radius.

To construct a static class diagram form the above problem statement, the steps are undertaken as follows:

- The first step in analyzing the problem statement is to identify a list of nouns and noun phrases, such as vessel, container, size, capacity, tank, rectangle, square, height, length and width.

- The next step is to take some of those nouns out of the list. Such nouns are those which are identical or they refer to the same thing. In the problem statement, tank, vessel and container are synonyms of the same thing. So vessel is picked up. Some nouns are not part of the problem although they are mentioned in the problem statement. A software engineer/practitioner must take them out of the list such as substance and liquid. In this problem, liquid is not part of the problem. Even if the software engineer/practitioner chooses not to take the liquid out. Such a class would be an isolated class in the static class diagram.

- All names that are generated from the previous step are arranged in an adjacent matrix as in figure 3.1.

- Fill up the table (below the diagonal) with letters (I, H, P), to have the letter I. The answer for the following question must be yes, Is "Left cell" a "top row"?. For example "Is a rectangular tank a vessel? ". The answer is yes. Therefore, the cell that corresponds to rectangular tank with vessel should have an "I".

- If the answer for a question is no, the cell remains empty. Like "Is a capacity a vessel?". The answer is no, therefore, the cell remains empty.

- Move to the next question which "is part-of" question. A question is generated to have an answer either yes or no. The question is "Has (left row) a (top cell) ?" or "Is the (left row) part of a (top cell)?". If the answer to any of them is yes, then the corresponding cell is marked with a "P".

- The next question to fill up the table is "Has (left row) a (top row)?". If the answer to this is yes, write an "H" in the cell corresponding to the "left row" and "top cell". In the example, some cells are changed to "H".

- The table now becomes like figure 3.1. Any row that has a P letter means that this is an attribute. A row which has a single "I" is a class. Others would be a class if the row is an adjective. That means it is an attribute. If it is a verb, then it is considered a method .

	Vessel	Capacity	Amount	Rectangular	Cubic tank	Cylindrical tank	Length	Height	Width	Radius	
Vessel	I										C
Capacity	P	I									A
Amount	P		I								A
Rectangular tank	I	H	H	I							C
Cubic tank	I	H	H	I	I						C
Cylindrical tank	I	H	H			I					C
Length				P	P		I				A
Height				P	P	P		I			A
Width				P					I		A
radius						P				I	A

Figure 3.1. Object-Oriented Relationship Information [1]

- With a pencil, form the English letter L by connecting I-cells together. You need to form a stair-shape so you can have the hierarchy form of the class diagram. The first path is (Vessel → Rectangular Tank, Rectangular Tank → Cubic Tank) and the next path is (Vessel → cylindrical Tank) as shown in figure 3.2.

- UML shapes of the hierarchy are drawn as shown in Figure 3.3 .

- Rearrange the table so that the classes are at the top and the attributes are at the bottom and apply the same for all of them as shown in Figure 3.2. The rearrangement should take into consideration the number of "I" cells within each row. The rows should be sorted in ascending order.

- For the "P" cells, form a region of one cell, two cells, three cells, four cells and so on.

- The two attributes capacity and amount can be in the top of the class hierarchy as you form a rectangular region of four cells and that region intersects with the top hierarchy which is vessel. This region covers the lower part of all L-shape.", such attribute must be at the top of the hierarchy.
- Length has two boxes and should be in the Rectangular tank class as in figure 3.2.

- Height has two regions, one with two boxes and the other with one as we can't form a region with three boxes that forms a lower part of L-shape that are related in the hierarchy. From this, one can notice that one is in the rectangular tank class and the other in the cylindrical tank class.

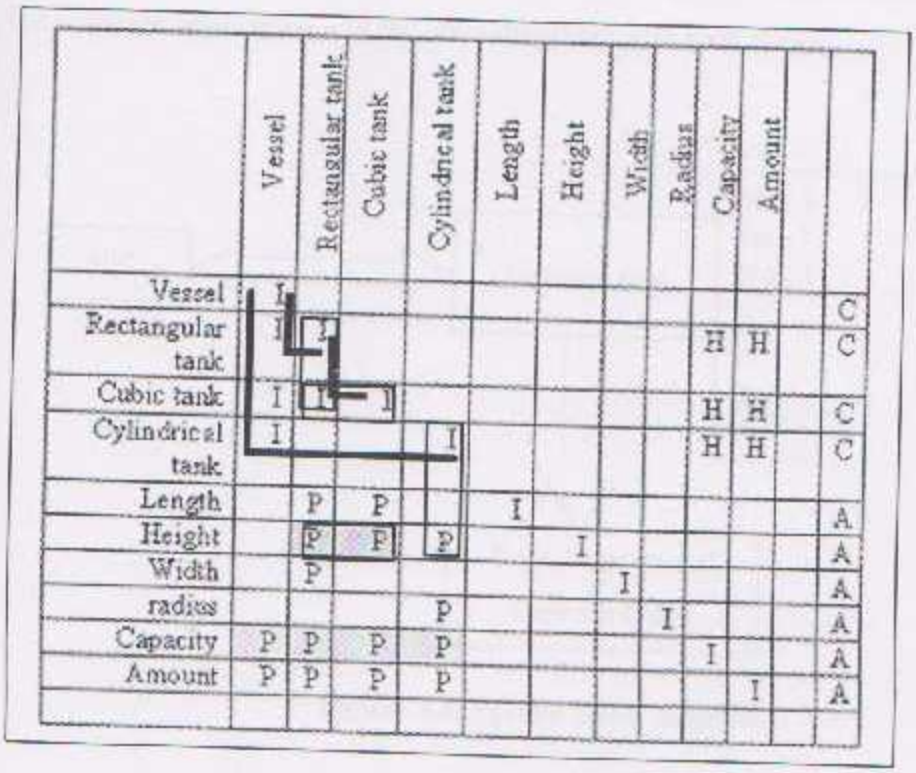

Figure 3.2. Class Relationship [1] .

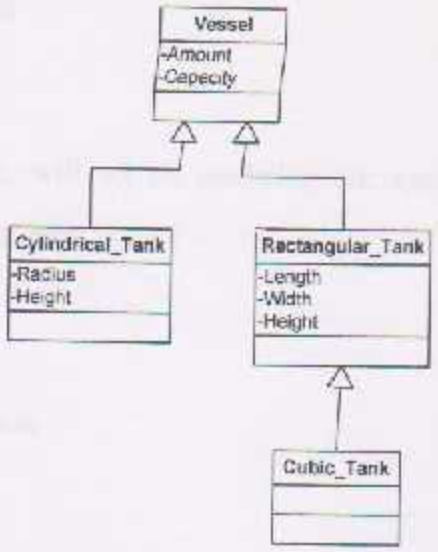

Figure 3.3. UML Static Class Diagram [1]

3.2.2 System's Block Diagram

The following figure shows our system's block diagram which consist of the user , the software application and the class diagrams with their codes.

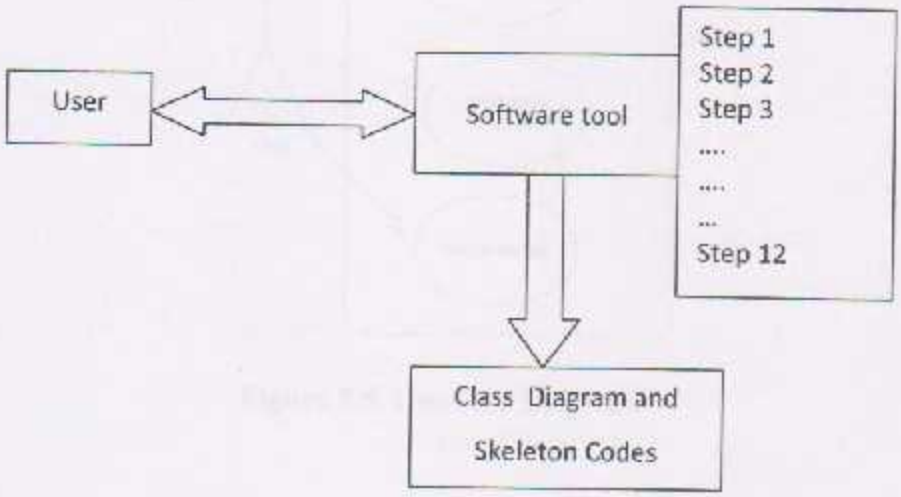

Figure 3.4. System block diagram.

3.3. System Modeling

In this section, will act on modeling our system by using Use Case and sequence modeling diagrams.

3.3.1. Use Case Diagram

A use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system . These diagrams are especially important in organizing and modeling the behaviors of a system .

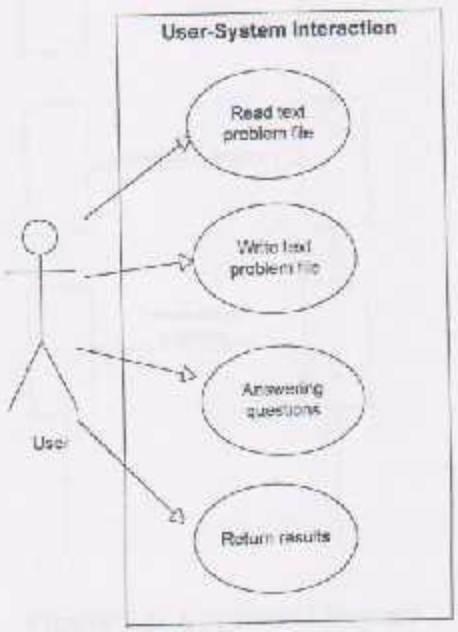

Figure 3.5. Use Case Diagram

3.3.2. Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages . Sequence diagram address the dynamic view of a system .

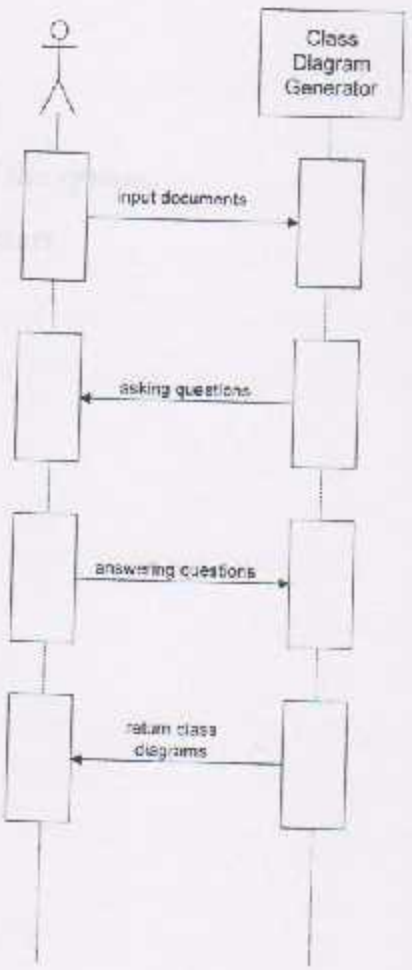

Figure 3.6. Sequence Diagram

Chapter 4

Detailed System Design

4.1. Design Options

4.2. Architecture of the system

4.3. System's Flowchart

4.4. Limitations

4.5. Summary

Chapter Four

Detailed System Design

In this chapter a detailed description of the design of different system components and modules, including system architecture, flow charts, GUI design, modules and specifications.

4.1. Design Options

In this section, we demonstrate design option for the system, and discuss why we choose it in implementing the system, in term of output format and programming language.

4.1.1 Output Format

In our system, we choose to design the output -which represent the class diagram of the input problem- as XML file besides a skeleton programming codes for these classes in VB.Net object-oriented programming language.

4.1.1.1. XML

XML (Extensible Markup Language) is a general-purpose specification for creating custom markup languages. It is classified as an extensible language, because it allows the user to define the mark-up elements. XML's purpose is to aid information systems in sharing structured data, especially via the Internet to encode documents, and to serialize data.

XML's set of tools helps developers in creating web pages but its usefulness goes well beyond that. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to reuse that content in other applications or for other presentation environments. Most importantly, XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion.[9,10]

XML began as a simplified subset of the Standard Generalized Markup Language (SGML), meant to be readable by people via semantic constraints; application languages can be implemented in XML. These include XHTML, RSS, MathML, GraphML, Scalable Vector Graphics, MusicXML, and others. Moreover, XML is sometimes used as the specification language for such application languages. XML is recommended by the World Wide Web Consortium (W3C). It is a fee-free open standard. [10]

Why XML ? (Advantages of XML)

- XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations. [10] XML data is stored in plain text format. This software- and hardware-independent way of storing data allows different incompatible systems to share data without needing to pass them through many layers of conversion. This also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing any data.
- With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities. [10]
- XML provides a gateway for communication between applications, even applications on wildly different systems. As long as applications can share data (through HTTP, file sharing, or another mechanism), and have an XML

parser, they can share structured information that is easily processed. Databases can trade tables, business applications can trade updates, and document systems can share information. [10]

- It supports Unicode, allowing almost any information in any written human language to be communicated.
- It can represent common computer science data structures: records, lists and trees.
- Its self-documenting format describes structure and field names as well as specific values.
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent.
- Content-based XML markup enhances searchability, making it possible for agents and search engines to categorize data instead of wasting processing power on context-based full-text searches. [10]
- XML is heavily used as a format for document storage and processing, both online and offline.
- It is based on international standards.
- It can be updated incrementally.
- It allows validation using schema languages such as XSD and Schematron, which makes effective unit-testing, firewalls, acceptance testing, contractual specification and software construction easier.
- It is platform-independent, thus relatively immune to changes in technology.
- Its predecessor, SGML, has been in use since 1986, so there is extensive experience and software available.

So we store the output of our application as XML file that represents the obtained class with their attributes and operations. Another XML file was generated to represent the whole problem from its input text and through its stored matrix to its outputs classes.

4.1.1.2. Classes Skeleton Programming Codes

We generate a skeleton codes of the obtained classes representing the classes , their data members (attributes) and their methods (operations). Also we represent the classes inheritance hierarchy and their constructors. All that was represented as VB.Net class files (the following section explains why we use VB.Net).

4.1.2 Visual Basic.NET Programming Language

We choose Visual Basic Dot Net as programming language for our application over other programming languages .

Visual Basic.Net is one the visual studio dot net programming languages which means that it has many features by its libraries and simplicity of using them. In other hand we used VB.Net over other Visual Studio languages like VC++ or C# because we have more experience and background in VB.Net than the others.

Other important reason of choosing VB.Net that it is a fully object- oriented construct as we dealing with classes to program in object- oriented languages . Also VB.Net provides the easiest, most productive language and tool for rapidly building windows and web applications. Visual Basic.NET comes with enhanced visual designers, increased application performance, easy portability and a powerful integrated development environment (IDE).

4.2. Architecture Of The System

The SCDC system consists of a Graphical User Interface (GUI) interacts with the user to generate the class diagram and the skeleton codes.

4.2.1.GUI Design

The following figures show the GUI of our application.

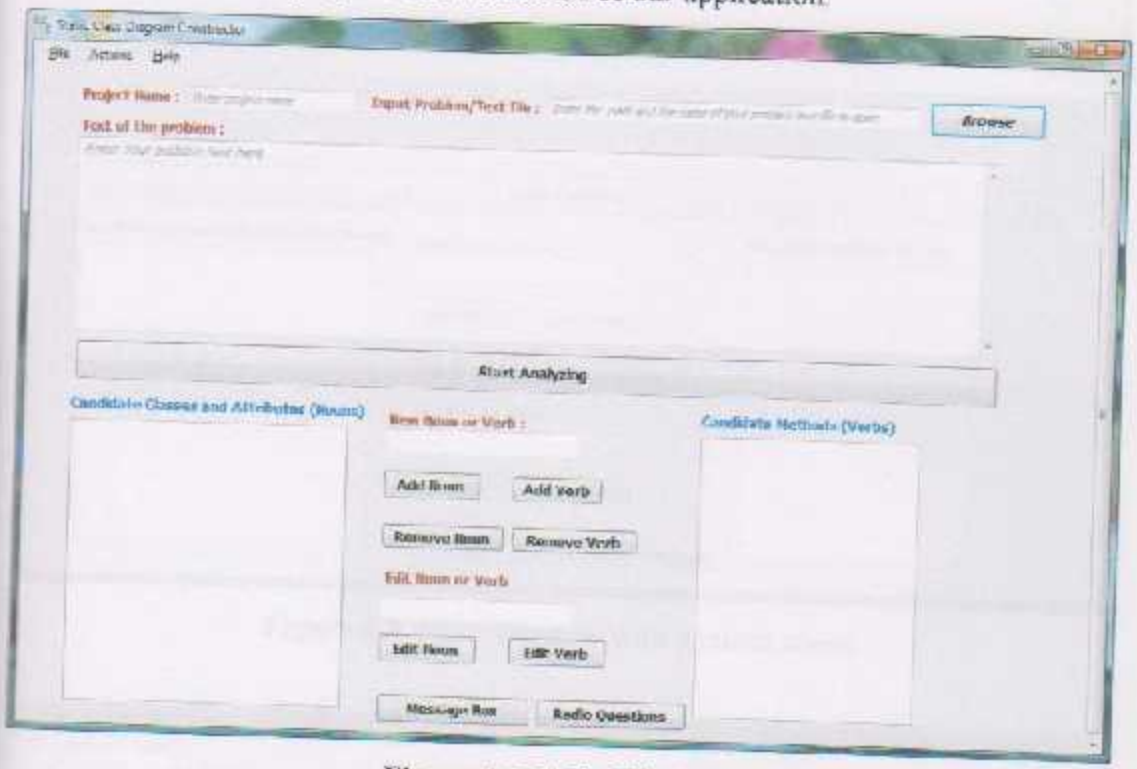

Figure 4.1. Main Window.

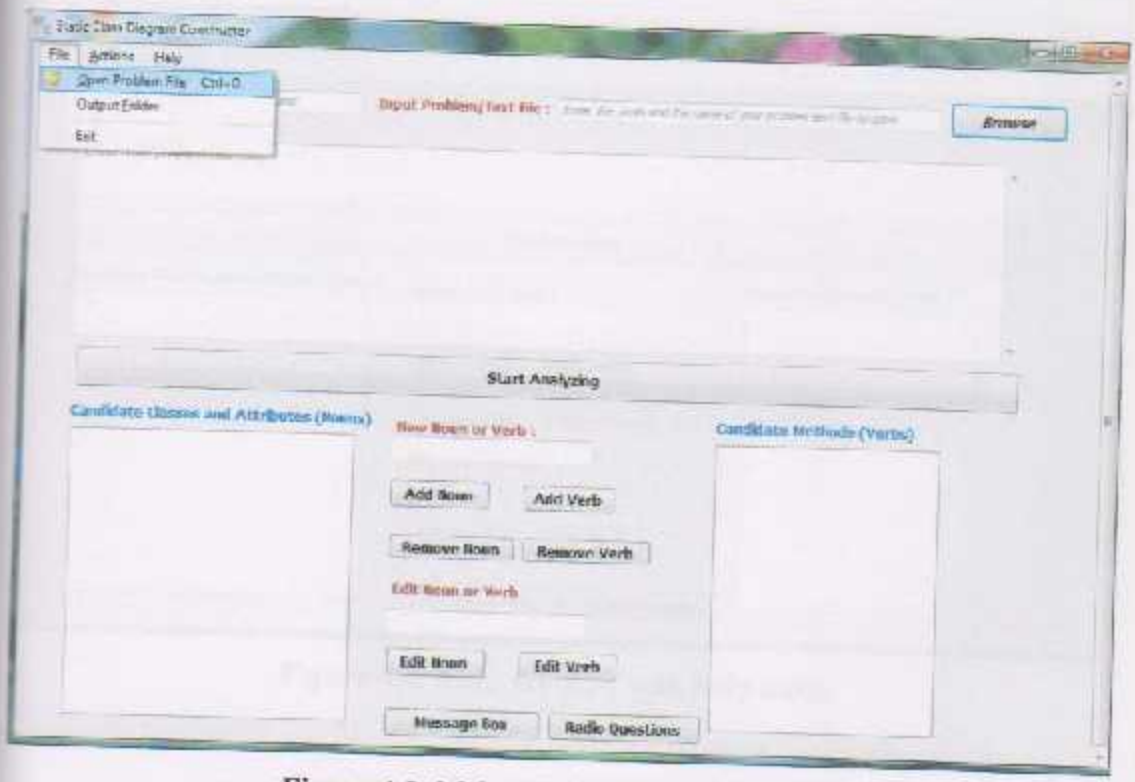

Figure 4.2. Main Window with File menu.

Figure 4.3. Main Window with Actions menu.

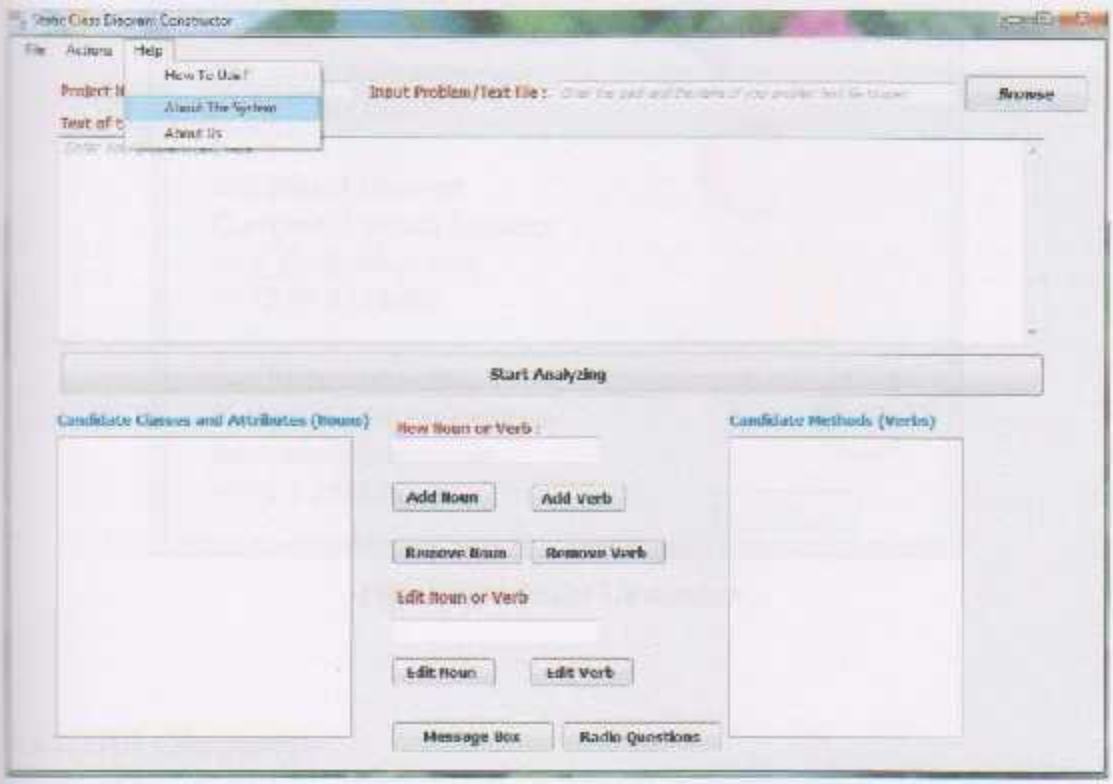

Figure 4.4. Main Window with Help menu.

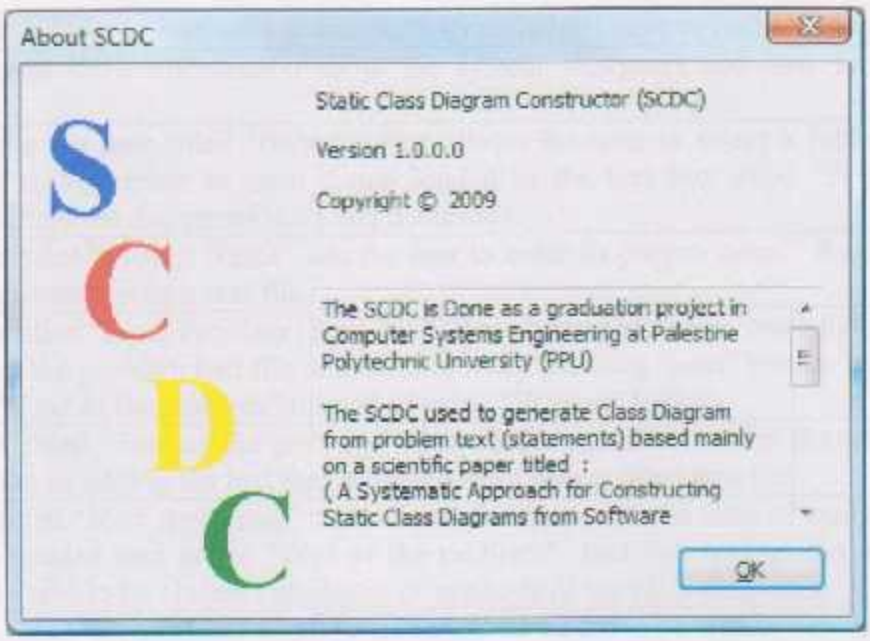

Figure 4.5. About The System window.

Figure 4.6. About Us window.

4.2.2. GUI Components

A File menu which allows the user to open problem text file , specify the output folder or exit from the application .
An Actions menu which allows to start analyzing of the problem text by finding

names and verbs or to start asking questions by message boxes or radio buttons.
A Help menu show information about the system ,designers and how to use this application.
A button for the user titled "Browse" that allows the user to select a folder of the problem text documents to open it and load it in the text box titled "Text of the problem" where the document is a (.txt) document.
A text box titled "Project Name" lets the user to enter its project name . By default it will be the name problem text file.
A text box titled "Input Problem /Text File" that allows the user to enter the path and the name of the problem text file and loading it -by pressing "load" button- in the text box titled "Text of the problem" instead of using "Browse" button.
A text box titled "Text of the problem" that allows the user to enter the text of the user problem or editing the text that is loaded from the problem text file.
A button titled "Start Analyzing" that allows the application to start of analyzing the entered or loaded text in the "Text of the problem" text box to find the candidate nouns and verbs to be classes , attributes or methods of the class diagrams.
A list box titled "Nouns List" displays all candidate nouns to be classes or attributes.
A list box titled "Verbs List" displays all candidate verbs to be methods.
A text box titled "New Noun or Verb" that allows the user to enter new noun or new verb to be added to the noun or verb list.
A button titled "Add Noun" that allows the user to add the new noun that is entered in the "New Noun or Verb" text box to the noun list.
A button titled "Add Verb" that allows the user to add the new verb that is entered in the "New Noun or Verb" text box to the verb list.
A button titled "Remove Noun" that allows the user to remove the selected noun from the noun list.
A button titled "Remove Verb" that allows the user to remove the selected verb from the verb list.
A text box titled "Edit Noun or Verb" that allows the user to edit the selected noun or verb in the noun or the verb list.
A button titled "Edit Noun" that allows the user to load the selected noun from the noun list into the "Edit Noun or Verb" text box to be edited .
A button titled "Edit Verb" that allows the user to load the selected verb from the verb list into the "Edited Noun or Verb" text box to be edited .
A button titled "Message Box Questions" that allows the application to start of asking the user of questions about the problem as message box for each question.
A button titled "Radio Questions" that allows the application to start of asking the user of questions about the problem as list of radio buttons.

4.3. System's Flowchart

In this section detailed functional description of different system components, and the functions contained within each component.

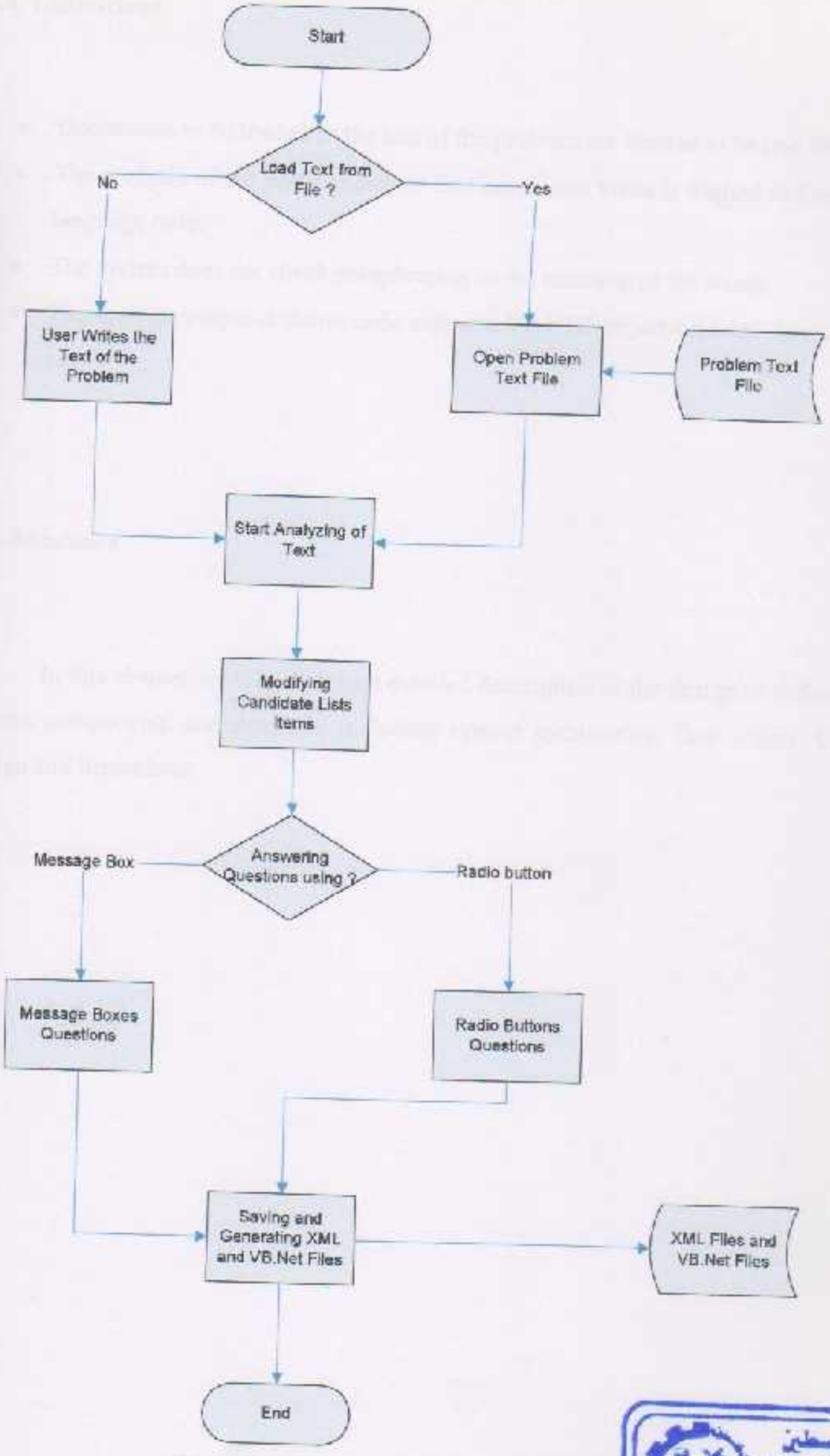

Figure 4.7. System's General Flow Chart.

4.4. Limitations

- Documents to be loaded as the text of the problem are limited to be text files.
- The analysis of the problem text to find nouns and verbs is limited to English language only.
- The system does not check paraphrasing or the meaning of the words.
- The systems output skeleton code supports VB.NET object-oriented language only .

4.5. Summary

In this chapter we talked about a detailed description of the design of different system components and modules, including system architecture, flow charts, GUI design and limitations.

Chapter 5

Implementation And Testing

Chapter Five

Implementation and Testing

In this chapter description of how the components of the system were implemented, and testing of the system.

5.1. Development Environment

The implementation of the system was done on Microsoft Visual Studio 2005 environment, using Visual Basic dot Net as programming language.

The software that was used in the implementation and testing phases consist of:

- Microsoft Visual Studio 2005 Professional Edition : very helpful IDE that provided us ready implemented GUI components that facilitated the design and implementation of the system, we used this IDE to design the GUI, and to write the source code of all functions.

- Microsoft dot NET Framework version 2.0 : should be installed on working computer, in order to compile to VB.Net source code and make it executable, also for the testing computers it should be installed as well, in order to run the application.

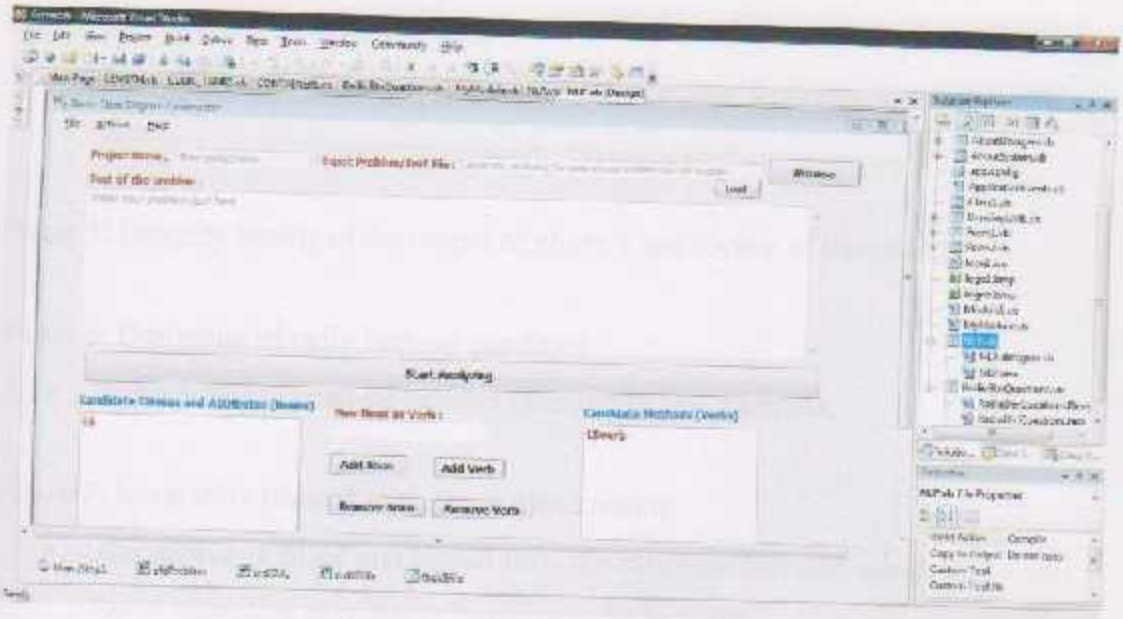

Figure 5.1. Visual Studio IDE

5.2. Development Process

The development process was done through the following phases :

Phase 1. Development of Natural Language Processing (NLP) :

- Implementation of an open source libraries that tokenize the input text and classify the tokens .
- Filter the NLP output to find the candidate classes names and their attributes and methods names and storing them into two lists.

Phase 2: Modifying the lists:

- Implementation of operations on the lists such as adding, editing or removing nouns or verbs form the lists.
- A final NLP out is ready in the lists.

Phase 3: setting up a message box module:

- A Yes / No questions message box is designed .
- The result of answering the questions is prepared to be stored in a matrix.

Phase 4: Integration of the output of phase 2 with phase 3:

- The matrix is filled and sorted then the relations between nouns and verbs found.

Phase 5: Integrity testing of the output of phase 4 and review of the system.

Phase 6: Designing of radio buttons questions:

- Radio buttons questions besides message boxes questions.

Phase 7: Integration phase 6 with phase 4 and testing.

- The matrix is filled and sorted then the relations between nouns and verbs found according to message boxes or radio buttons.

Phase 8: Designing and implantation of the XML files.

- The classes and their attributes, operations and relations is written into XML file.
- The whole project from text problem and through the matrix and to the obtained classes is stored as another XML file.

Phase 9: Integrity testing of the output of phase 8 and review the system.

Phase 10: Expansion of the system to generate skeleton codes

- Skeleton programming codes in VB.Net for the obtained classes is generated and stored as VB.Net programming files (.vb).

Phase 11: Intensive integrity testing of the system and delivery of the system.

Component testing was done during each phase that included implementing of any component of the system.

5.3. Testing

In this section we showed the error messages for invalid inputs to the system and demonstration of an example of using the system.

5.3.1. Error messages

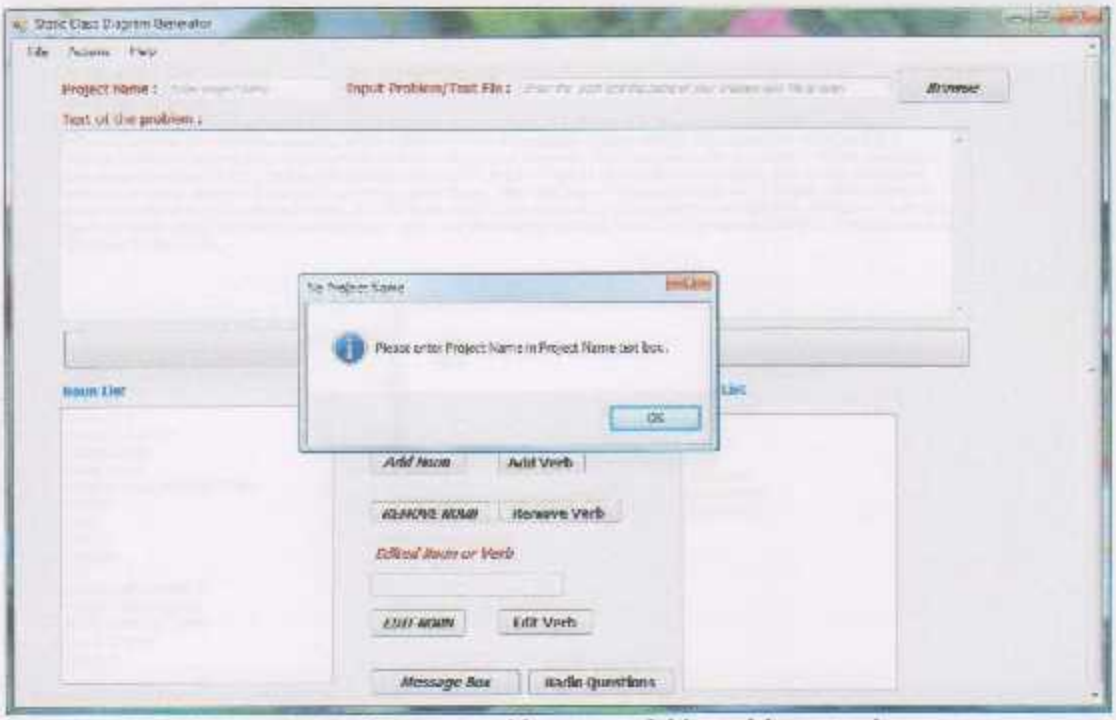

Figure 5.2. Trying to specify output folder with no project.

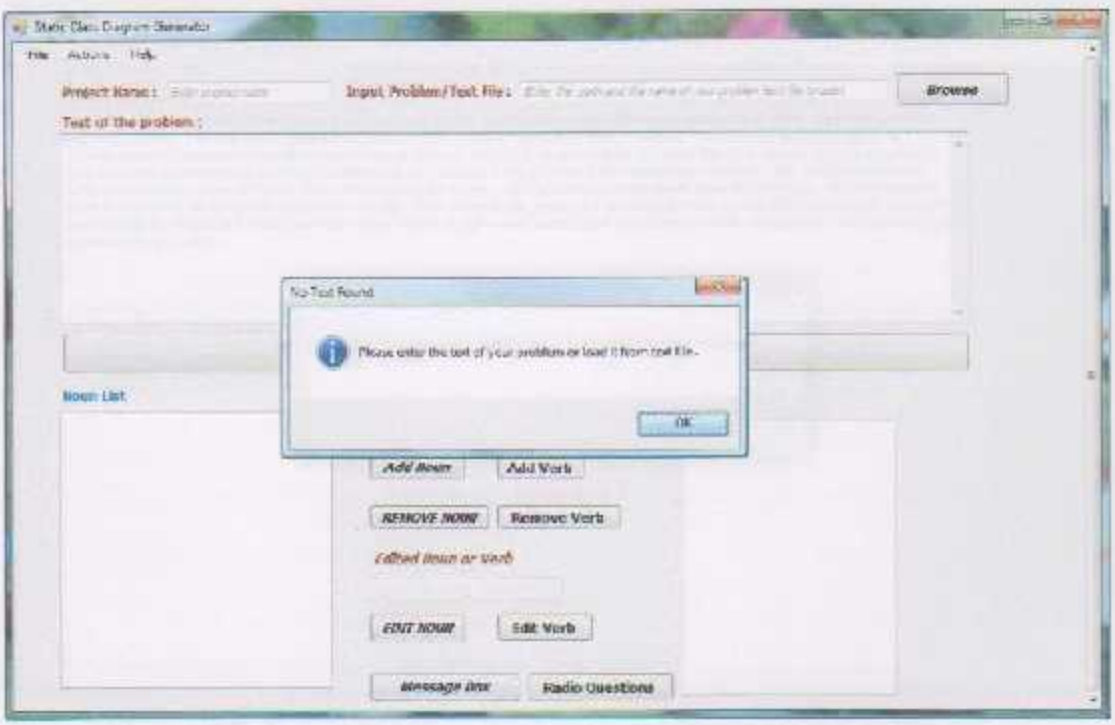

Figure 5.3. Trying to start analysis with no problem text .

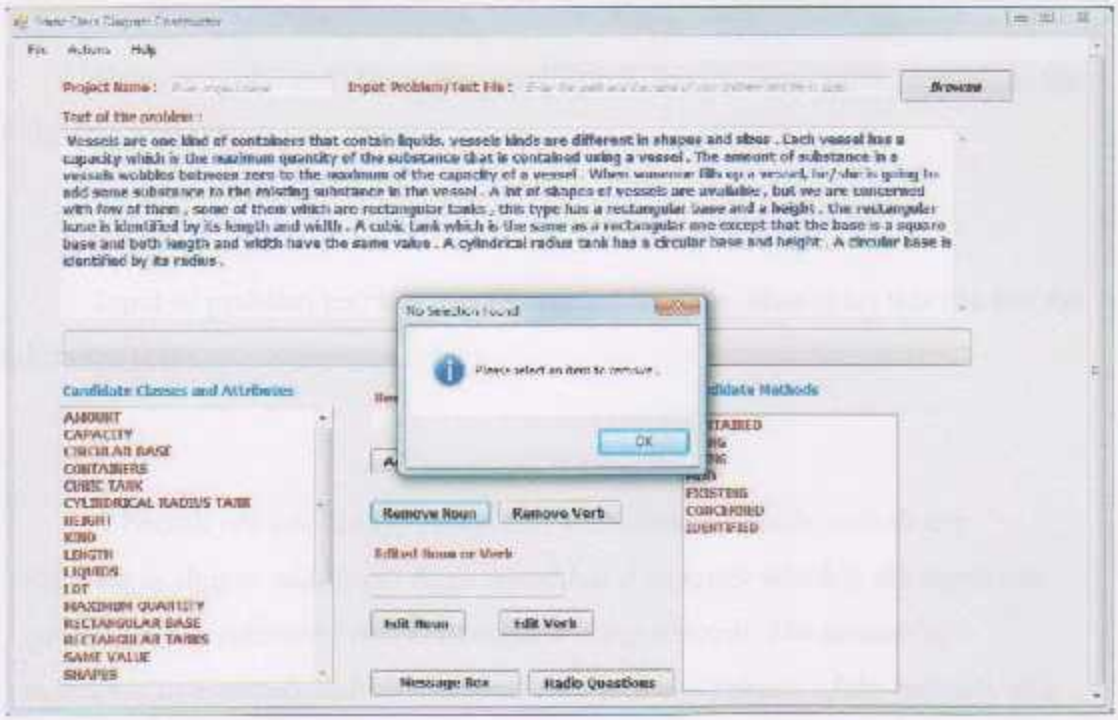

Figure 5.4. Trying to remove an item with no selection.

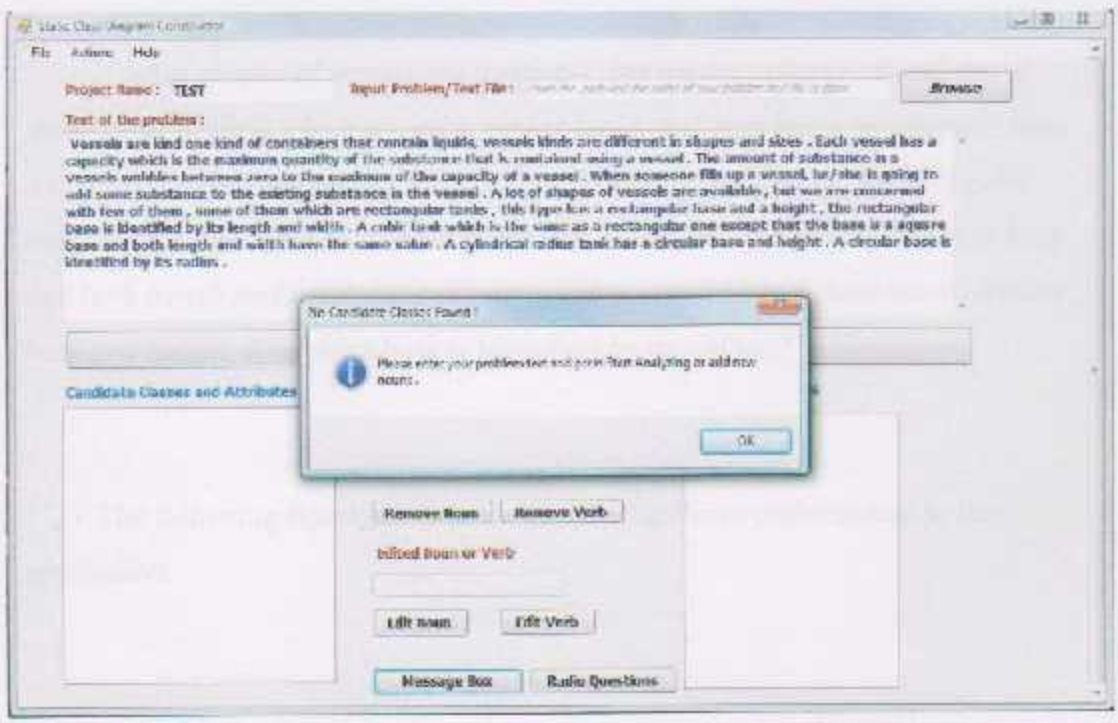

Figure 5.5. Trying to answering questions without candidate classes.

5.3.2. Testing Example

Here we showed how the application interaction when applying the following example:

Input of problem text is a text file named Vessels_Shapes.txt this file has the following text :

" Vessels are one kind of containers that contain liquids, vessels are different in shapes and sizes. Each vessel has a capacity which is the maximum quantity of the substance that is contained using a vessel. The amount of substance in a vessels wobbles between zero to the maximum of the capacity of a vessel. When someone fills up a vessel, he/she is going to add some substance to the existing substance in the vessel.

A lot of shapes of vessels are available, but we are concerned with few of them, some of them which are rectangular tanks, this type has a rectangular base and a height, the rectangular base is identified by its length and width. A cubic tank which is the same as a rectangular one except that the base is a square base and both length and width have the same value. A cylindrical tank has a circular base and height. A circular base is identified by its radius ."

The following figure show how to apply the above problem text to the application.

- Click "Browse" button or from file menu to open and load problem text as Figure 5.6.

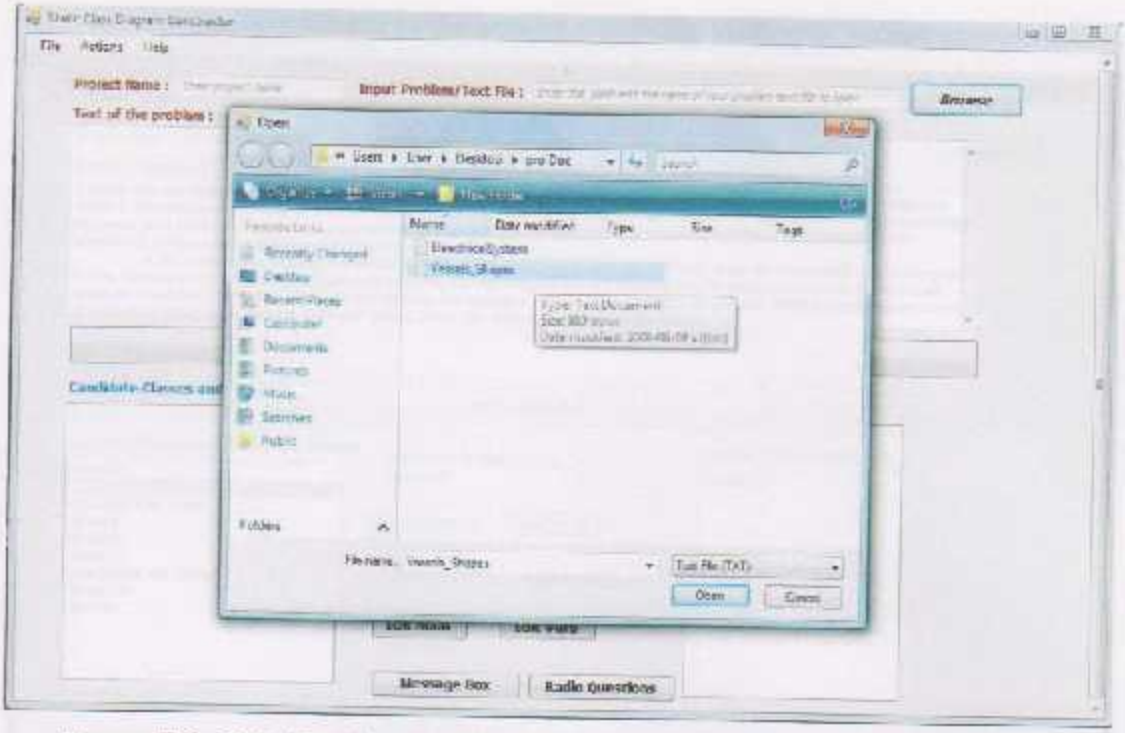

Figure 5.6. GUI: Opening Vessels_Shapes.txt file as the input problem file.

- Click "Start Analyzing" from Actions menu or "Start Analyzing" button.

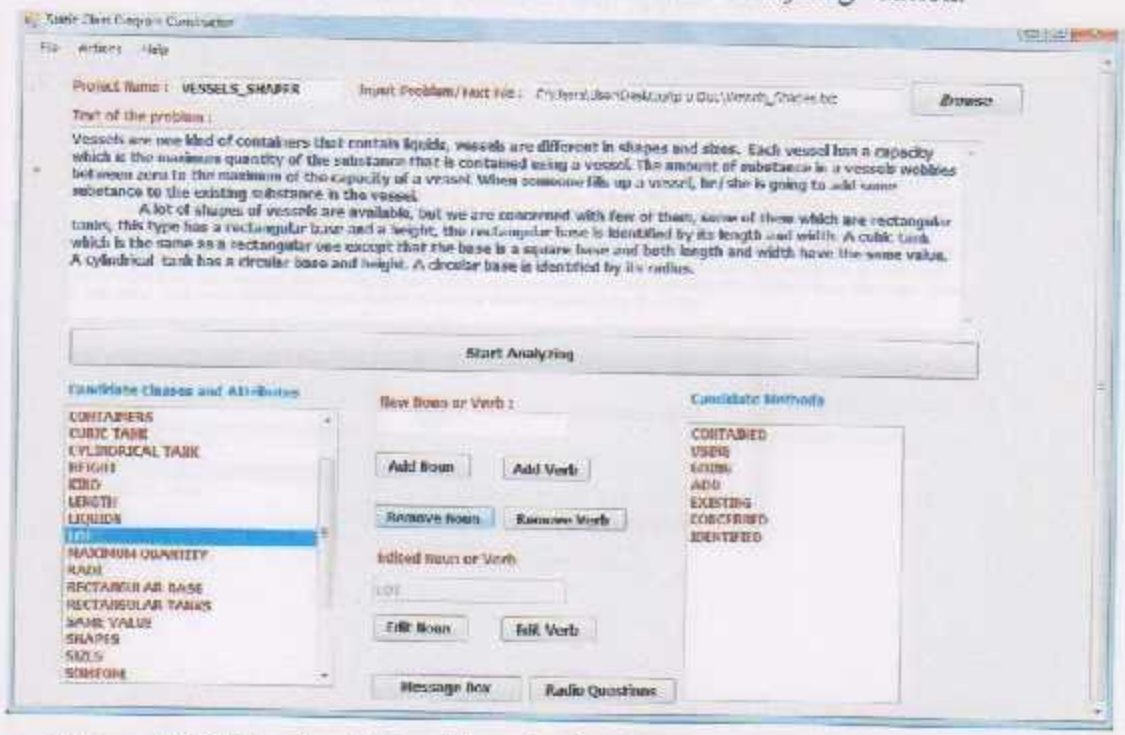

Figure 5.7. GUI: after hitting "Start Analyzing" from Actions menu or "Start Analyzing" button.

- Modify the candidate items in the lists by removing, editing or adding new items.

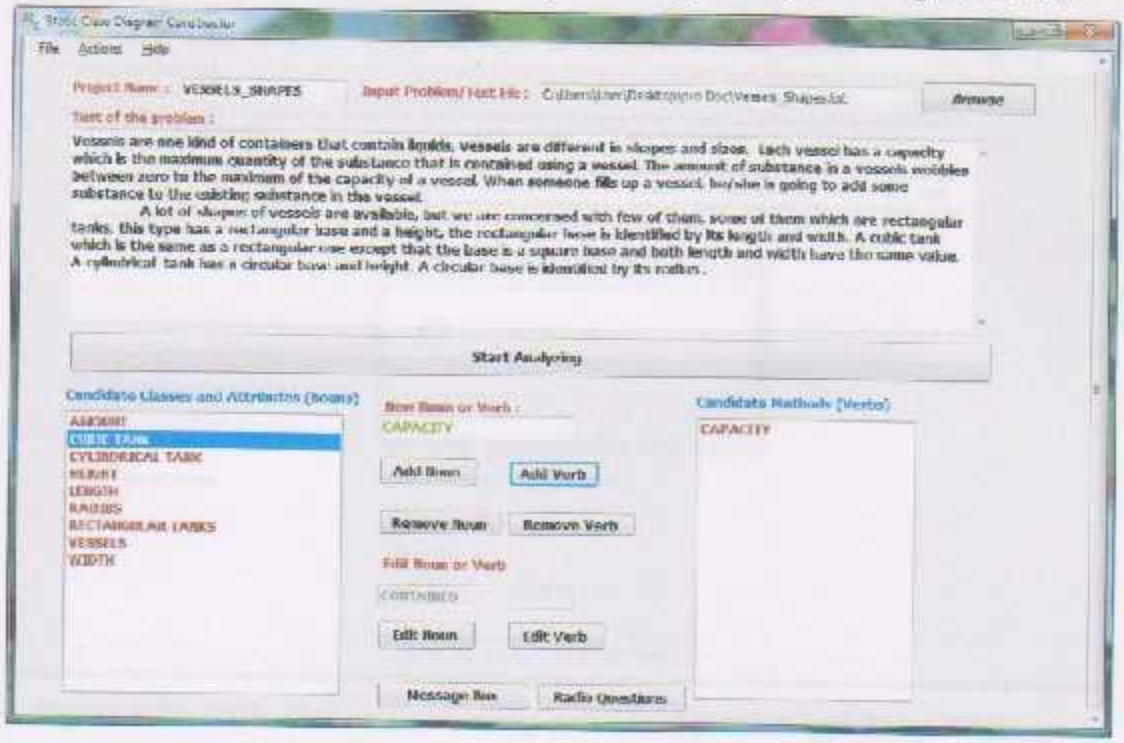

Figure 5.8. GUI: after modifying the lists of candidate classes and methods.

- Click answering questions from actions menu and choose "Message Box" or "Radio Buttons" and a series of question will appear like the following figures.

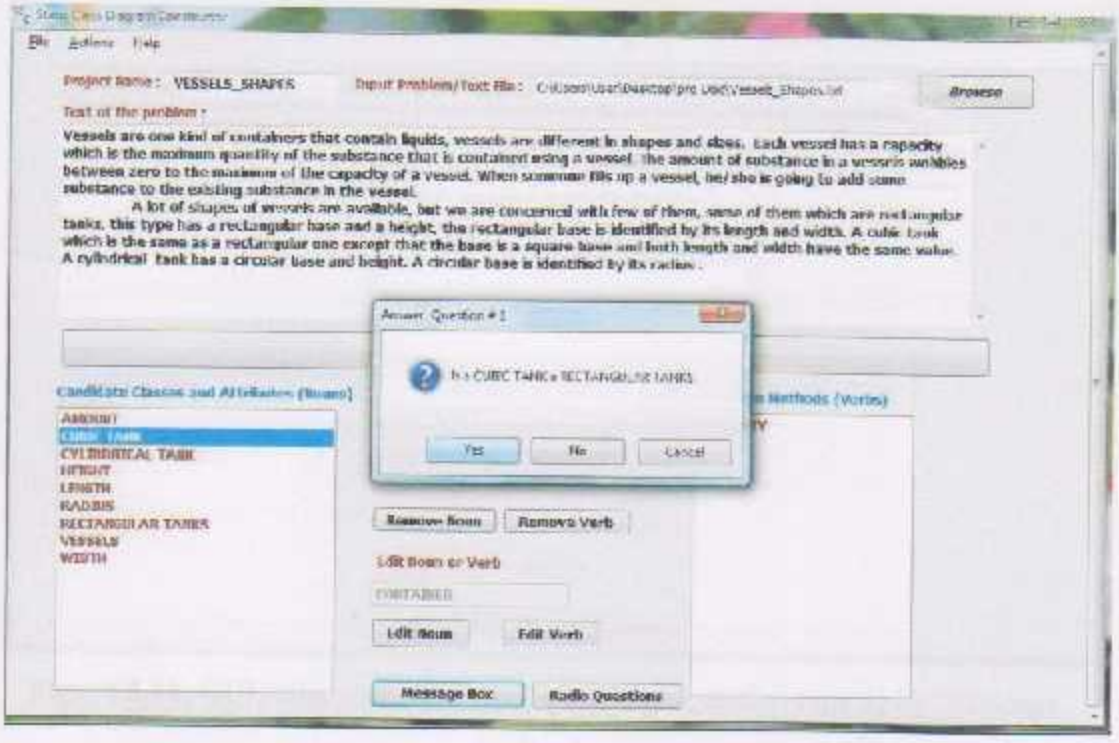

Figure 5.9. GUI: after start answering questions (question type 1) by "Message Box".

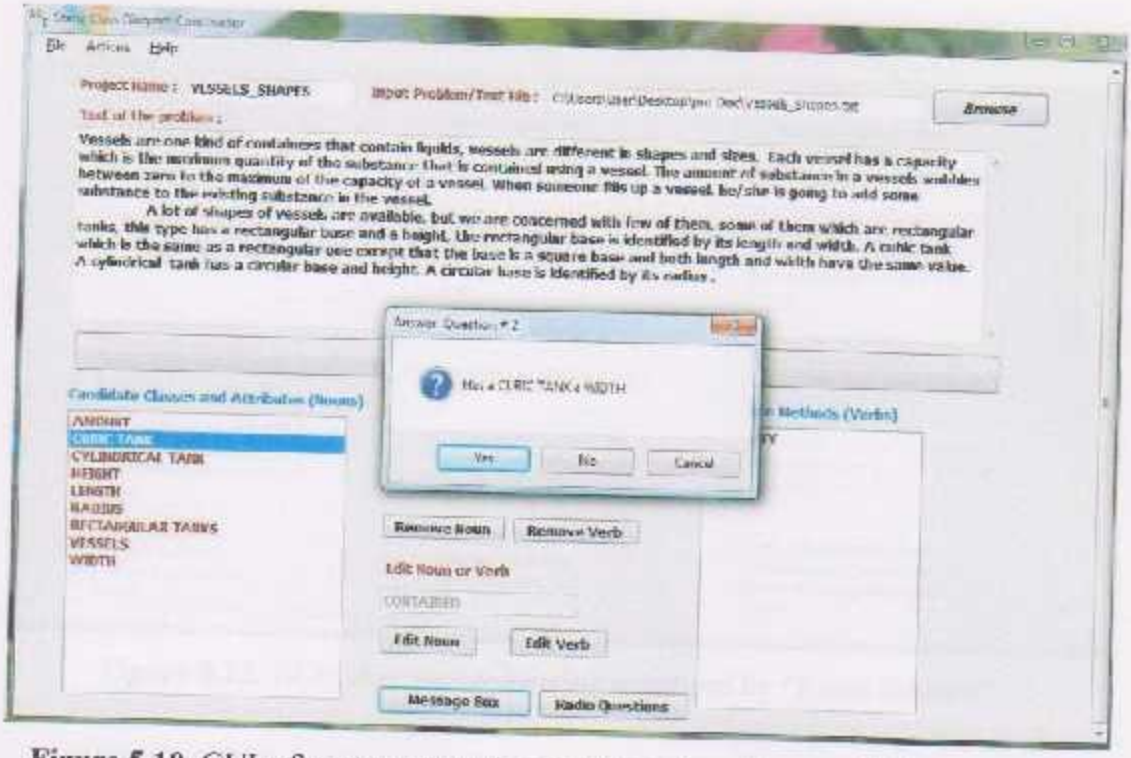

Figure 5.10. GUI: after start answering questions (question type 2) by "Message Box".

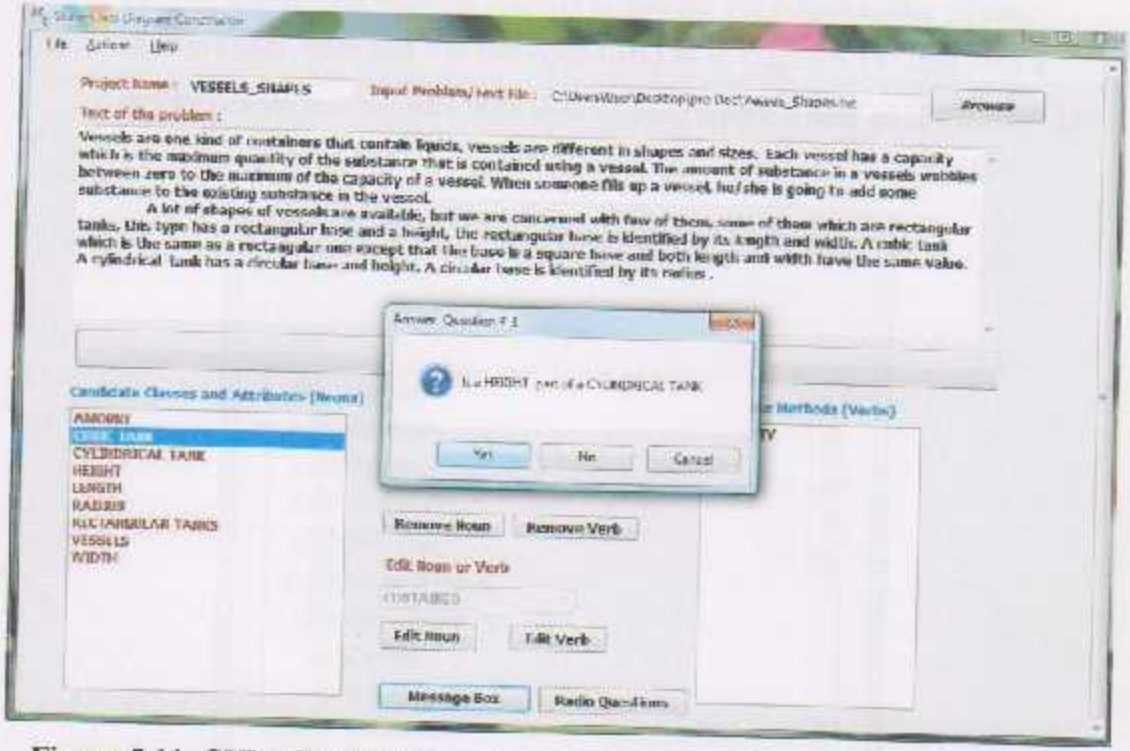

Figure 5.11. GUI: after start answering questions (question type 3) by "Message Box".

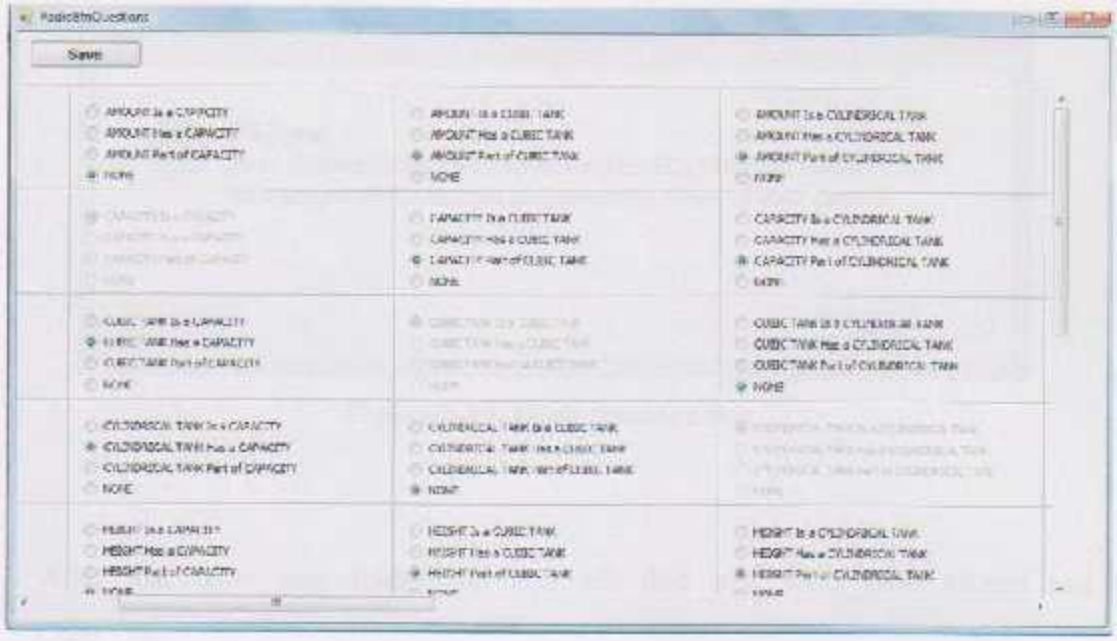

Figure 5.12. GUI: after start answering questions by "Radio Buttons".

- After answering all questions in message boxes or by clicking save in radio buttons questions a window appears to specify the output folder of your classes (XML files and VB.Net code files) as in Figure 5.13. after than a done message box appears with path of your output folder as in Figure 5.14

Figure 5.13. Specifying the output folder.

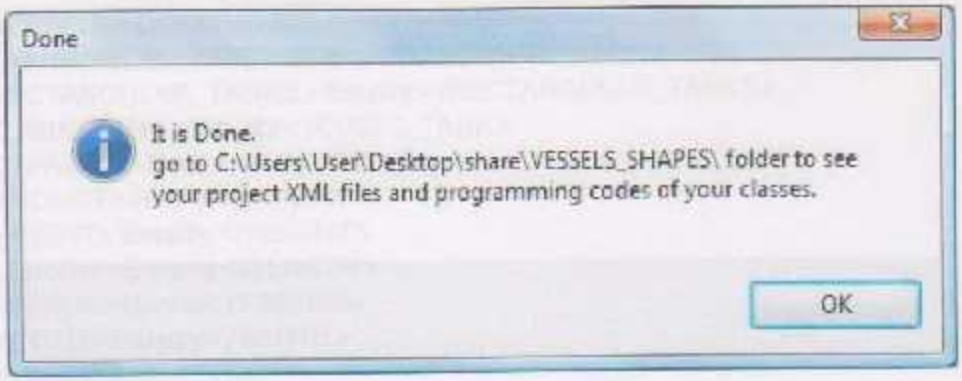

Figure 5.14. Done message box.

- After that open your folder and you will find two XML files All.xml and Class.xml .

All.xml File

```
<?xml version="1.0" ?>
- <Project>
  <ProjectName>VESSELS_SHAPES</ProjectName>
  <ProblemText>Vessels are one kind of containers that contain liquids, vessels are different in shapes and sizes. Each vessel has a capacity which is the maximum quantity of the substance that is contained using a vessel. The amount of substance in a vessels wobbles between zero to the maximum of the capacity of a vessel. When someone fills up a vessel, he/she is going to add some substance to the existing substance in the vessel. A lot of shapes of vessels are available, but we are concerned with few of them, some of them which are rectangular tanks, this type has a rectangular base and a height, the rectangular base is identified by its length and width. A cubic tank which is the same as a rectangular one except that the base is a square base and both length and width have the same value. A cylindrical tank has a circular base and height. A circular base is identified by its radius .</ProblemText>
- <Matrix>
- <VESSELS>
  <VESSELS>Is a</VESSELS>
  <CYLINDRICAL_TANK>Empty</CYLINDRICAL_TANK>
  <RECTANGULAR_TANKS>Empty</RECTANGULAR_TANKS>
  <CUBIC_TANK>Empty</CUBIC_TANK>
  <CAPACITY>Has a</CAPACITY>
  <AMOUNT>Has a</AMOUNT>
  <HEIGHT>Empty</HEIGHT>
  <LENGTH>Empty</LENGTH>
  <RADIUS>Empty</RADIUS>
  <WIDTH>Empty</WIDTH>
  <Type>Class</Type>
  </VESSELS>
- <CYLINDRICAL_TANK>
```

<VESSELS>**Is a**</VESSELS>
<CYLINDRICAL_TANK>**Is a**</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>**Empty**</RECTANGULAR_TANKS>
<CUBIC_TANK>**Empty**</CUBIC_TANK>
<CAPACITY>**Has a**</CAPACITY>
<AMOUNT>**Has a**</AMOUNT>
<HEIGHT>**Empty**</HEIGHT>
<LENGTH>**Empty**</LENGTH>
<RADIUS>**Has a**</RADIUS>
<WIDTH>**Empty**</WIDTH>
<Type>**Class**</Type>
</CYLINDRICAL_TANK>
- <RECTANGULAR_TANKS>
<VESSELS>**Is a**</VESSELS>
<CYLINDRICAL_TANK>**Empty**</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>**Is a**</RECTANGULAR_TANKS>
<CUBIC_TANK>**Empty**</CUBIC_TANK>
<CAPACITY>**Has a**</CAPACITY>
<AMOUNT>**Has a**</AMOUNT>
<HEIGHT>**Has a**</HEIGHT>
<LENGTH>**Has a**</LENGTH>
<RADIUS>**Empty**</RADIUS>
<WIDTH>**Has a**</WIDTH>
<Type>**Class**</Type>
</RECTANGULAR_TANKS>
- <CUBIC_TANK>
<VESSELS>**Is a**</VESSELS>
<CYLINDRICAL_TANK>**Empty**</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>**Is a**</RECTANGULAR_TANKS>
<CUBIC_TANK>**Is a**</CUBIC_TANK>
<CAPACITY>**Has a**</CAPACITY>
<AMOUNT>**Has a**</AMOUNT>
<HEIGHT>**Has a**</HEIGHT>
<LENGTH>**Has a**</LENGTH>
<RADIUS>**Empty**</RADIUS>
<WIDTH>**Has a**</WIDTH>
<Type>**Class**</Type>
</CUBIC_TANK>
- <CAPACITY>
<VESSELS>**Part of**</VESSELS>
<CYLINDRICAL_TANK>**Part of 2**</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>**Part of 2**</RECTANGULAR_TANKS>
<CUBIC_TANK>**Part of 2**</CUBIC_TANK>
<CAPACITY>**Is a**</CAPACITY>
<AMOUNT>**Empty**</AMOUNT>
<HEIGHT>**Empty**</HEIGHT>
<LENGTH>**Empty**</LENGTH>
<RADIUS>**Empty**</RADIUS>
<WIDTH>**Empty**</WIDTH>
<Type>**Method**</Type>
</CAPACITY>
- <AMOUNT>

```
<VESSELS>Part of</VESSELS>
<CYLINDRICAL_TANK>Part of 2</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>Part of 2</RECTANGULAR_TANKS>
<CUBIC_TANK>Part of 2</CUBIC_TANK>
<CAPACITY>Empty</CAPACITY>
<AMOUNT>Is a</AMOUNT>
<HEIGHT>Empty</HEIGHT>
<LENGTH>Empty</LENGTH>
<RADIUS>Empty</RADIUS>
<WIDTH>Empty</WIDTH>
<Type>Attrib</Type>
</AMOUNT>
- <HEIGHT>
<VESSELS>Empty</VESSELS>
<CYLINDRICAL_TANK>Part of</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>Part of</RECTANGULAR_TANKS>
<CUBIC_TANK>Part of 2</CUBIC_TANK>
<CAPACITY>Empty</CAPACITY>
<AMOUNT>Empty</AMOUNT>
<HEIGHT>Is a</HEIGHT>
<LENGTH>Empty</LENGTH>
<RADIUS>Empty</RADIUS>
<WIDTH>Empty</WIDTH>
<Type>Attrib</Type>
</HEIGHT>
- <LENGTH>
<VESSELS>Empty</VESSELS>
<CYLINDRICAL_TANK>Empty</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>Part of</RECTANGULAR_TANKS>
<CUBIC_TANK>Part of 2</CUBIC_TANK>
<CAPACITY>Empty</CAPACITY>
<AMOUNT>Empty</AMOUNT>
<HEIGHT>Empty</HEIGHT>
<LENGTH>Is a</LENGTH>
<RADIUS>Empty</RADIUS>
<WIDTH>Empty</WIDTH>
<Type>Attrib</Type>
</LENGTH>
- <RADIUS>
<VESSELS>Empty</VESSELS>
<CYLINDRICAL_TANK>Part of</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>Empty</RECTANGULAR_TANKS>
<CUBIC_TANK>Empty</CUBIC_TANK>
<CAPACITY>Empty</CAPACITY>
<AMOUNT>Empty</AMOUNT>
<HEIGHT>Empty</HEIGHT>
<LENGTH>Empty</LENGTH>
<RADIUS>Is a</RADIUS>
<WIDTH>Empty</WIDTH>
<Type>Attrib</Type>
</RADIUS>
- <WIDTH>
```

```
<VESSELS>Empty</VESSELS>
<CYLINDRICAL_TANK>Empty</CYLINDRICAL_TANK>
<RECTANGULAR_TANKS>Part of</RECTANGULAR_TANKS>
<CUBIC_TANK>Part of 2</CUBIC_TANK>
<CAPACITY>Empty</CAPACITY>
<AMOUNT>Empty</AMOUNT>
<HEIGHT>Empty</HEIGHT>
<LENGTH>Empty</LENGTH>
<RADIUS>Empty</RADIUS>
<WIDTH>Is a</WIDTH>
<Type>Attrib</Type>
</WIDTH>
</Matrix>
- <UML>
- <Package>
<ProjectName>VESSELS_SHAPES</ProjectName>
- <Class>
<ClaasName>VESSELS</ClaasName>
- <Attributes>
<Item>AMOUNT</Item>
</Attributes>
- <Operations>
<Item>CAPACITY</Item>
</Operations>
<Super>Object</Super>
- <Sub>
<Item>CYLINDRICAL_TANK</Item>
<Item>RECTANGULAR_TANKS</Item>
</Sub>
</Class>
- <Class>
<ClaasName>CYLINDRICAL_TANK</ClaasName>
- <Attributes>
<Item>HEIGHT</Item>
<Item>RADIUS</Item>
</Attributes>
<Operations />
<Super>VESSELS</Super>
<Sub />
</Class>
- <Class>
<ClaasName>RECTANGULAR_TANKS</ClaasName>
- <Attributes>
<Item>HEIGHT</Item>
<Item>LENGTH</Item>
<Item>WIDTH</Item>
</Attributes>
<Operations />
<Super>VESSELS</Super>
- <Sub>
<Item>CUBIC_TANK</Item>
</Sub>
```

```
  </Class>
- <Class>
  <ClaasName>CUBIC_TANK</ClaasName>
  <Attributes />
  <Operations />
  <Super>RECTANGULAR_TANKS</Super>
  <Sub />
  </Class>
  </Package>
  </UML>
  </Project>
```

Class.xml File

```
<?xml version="1.0" ?>
- <Package>
  <ProjectName>VESSELS_SHAPES</ProjectName>
- <Class>
  <ClaasName>VESSELS</ClaasName>
- <Attributes>
  <Item>AMOUNT</Item>
  </Attributes>
- <Operations>
  <Item>CAPACITY</Item>
  </Operations>
  <Super>Object</Super>
- <Sub>
  <Item>CYLINDRICAL_TANK</Item>
  <Item>RECTANGULAR_TANKS</Item>
  </Sub>
  </Class>
- <Class>
  <ClaasName>CYLINDRICAL_TANK</ClaasName>
- <Attributes>
  <Item>HEIGHT</Item>
  <Item>RADIUS</Item>
  </Attributes>
  <Operations />
  <Super>VESSELS</Super>
  <Sub />
  </Class>
- <Class>
  <ClaasName>RECTANGULAR_TANKS</ClaasName>
- <Attributes>
```

```
<Item>HEIGHT</Item>
<Item>LENGTH</Item>
<Item>WIDTH</Item>
</Attributes>
<Operations />
<Super>VESSELS</Super>
- <Sub>
<Item>CUBIC_TANK</Item>
</Sub>
</Class>
- <Class>
<ClaasName>CUBIC_TANK</ClaasName>
<Attributes />
<Operations />
<Super>RECTANGULAR_TANKS</Super>
<Sub />
</Class>
</Package>
```

Also you will find a folder named VB_Code and you will find four VB.Net code files in it (VESSELS.vb, RECTANGULAR_TANKS.vb, CYLINDRICAL_TANK.vb and CUBIC_TANK.vb file)

VESSELS.vb File

```
' This is done by Ibrahin ,Sari and Sari

' You can write Imports statements here if you need

 ' This class under project called : VESSELS_SHAPES
' You can change Public to privite or to other if you need
Public Class VESSELS

Inherits Object

' Declaration of the class attributes
Dim ValueA as New Object ' Enter your type here insted of Object
type.
Dim AMOUNT as New Object ' Enter your type here insted of Object
type.

' Class Constructors
Public Sub New()

'any other code if needed
```

```
End Sub 'end new

Public Sub New(byVal ValueA as Object ) ' Enter your type here
insted of Object type.
me.ValueA = ValueA

'any other code if needed


End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal AMOUNT as Object ) '
Enter your type here insted of Object type.
me.ValueA = ValueA
me.AMOUNT = AMOUNT

'any other code if needed


End Sub 'end new


'The Class Operations (methods)
Public Sub ValueM( )
 ' write your code for this method here if you need
' you can modify this method to pass values to it if you need


End Sub 'end of ValueM

Public Sub CAPACITY( )
 ' write your code for this method here if yiu need
' you can modify this method to pass values to it if you need


End Sub 'end of CAPACITY


End Class
```

CYLINDRICAL_TANK.vb File

```
' This is done by Ibrahim ,Sari and Sami

' You can write Imports statements here if you need

 ' This class under project called : VESSELS_SHAPES
' You can change Public to privite or to other if you need
Public Class CYLINDRICAL_TANK

Inherits VESSELS


' Declaration of the class attributes
Dim ValueA as New Object ' Enter your type here insted of Object
```

```
type.
Dim HEIGHT as New Object ' Enter your type here insted of Object
type.
Dim RADIUS as New Object ' Enter your type here insted of Object
type.


' Class Constructors
Public Sub New()

'any other code if needed

End Sub 'end new

Public Sub New(byVal ValueA as Object ) ' Enter your type here
insted of Object type.
me.ValueA = ValueA

'any other code if needed


End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal HEIGHT as Object ) '
Enter your type here insted of Object type.
me.ValueA = ValueA
me.HEIGHT = HEIGHT

'any other code if needed


End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal HEIGHT as Object ,
byVal RADIUS as Object ) ' Enter your type here insted of Object
type.
me.ValueA = ValueA
me.HEIGHT = HEIGHT
me.RADIUS = RADIUS

'any other code if needed


End Sub 'end new



''The Class Operations (methods)
Public Sub ValueM( )
 ' write your code for this method here if you need
' you can modify this method to pass values to it if you need


End Sub 'end of ValueM

End Class
```

RECTANGULAR_TANKS.vb File

```
' This is done by Ibrahim ,Sari and Sari

' You can write Imports statements here if you need

 ' This class under project called : VESSELS_SHAPES
' You can change Public to privite or to other if you need
Public Class RECTANGULAR_TANKS

Inherits VESSELS


' Declaration of the class attributes
Dim ValueA as New Object ' Enter your type here insted of Object
type.
Dim HEIGHT as New Object ' Enter your type here insted of Object
type.
Dim LENGTH as New Object ' Enter your type here insted of Object
type.
Dim WIDTH as New Object ' Enter your type here insted of Object
type.


' Class Constructors
Public Sub New()

'any other code if needed

End Sub 'end new

Public Sub New(byVal ValueA as Object ) ' Enter your type here
insted of Object type.
me.ValueA = ValueA

'any other code if needed


End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal HEIGHT as Object ) '
Enter your type here insted of Object type.
me.ValueA = ValueA
me.HEIGHT = HEIGHT

'any other code if needed


End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal HEIGHT as Object ,
byVal LENGTH as Object ) ' Enter your type here insted of Object
type.
me.ValueA = ValueA
me.HEIGHT = HEIGHT
me.LENGTH = LENGTH

'any other code if needed
```

```
End Sub 'end new

Public Sub New(byVal ValueA as Object , byVal HEIGHT as Object ,
byVal LENGTH as Object , byVal WIDTH as Object ) ' Enter your type
here insted of Object type.
me.ValueA = ValueA
me.HEIGHT = HEIGHT
me.LENGTH = LENGTH
me.WIDTH = WIDTH

'any other code if needed


End Sub 'end new



'The Class Operations (methods)
Public Sub ValueM( )
 ' write your code for this method here if you need
' you can modify this method to pass values to it if you need


End Sub 'end of ValueM


End Class
```

CUBIC_TANK.vb File

```
' This is done by Ibrahim ,Sari and Sari

' You can write Imports statements here if you need

 ' This class under project called : VESSELS_SHAPES
' You can change Public to privite or to other if you need
Public Class CUBIC_TANK

Inherits RECTANGULAR_TANKS


' Declaration of the class attributes
Dim ValueA as New Object ' Enter your type here insted of Object
type.


' Class Constructors
Public Sub New()

'any other code if needed

End Sub 'end new

Public Sub New(byVal ValueA as Object ) ' Enter your type here
```

```
insted of Object type.
me.ValueA = ValueA

'any other code if needed


End Sub 'end new



'The Class Operations (methods)
Public Sub ValueM( )
  ' write your code for this method here if yiu need
' you can modify this method to pass values to it if you need


End Sub 'end of ValueM


End Class
```

5.3. Summary

In this chapter description of how the components of the system were implemented, testing of the system, applying example, development process and phases.

Chapter 6

Conclusions And Future Work

Chapter Six

Conclusions and Future Work

In this chapter a description of the conclusions concluded from working on the project, future work and improvements to be done on the system.

6.1. Conclusions

From working on this project the following is concluded:

- Static class diagram, their relations and their programming codes is obtained using this application.
- Using this application finding static class diagram, their relations and their programming code become simpler and easier. Also it saves the time of the user .
- Team work is very effective in large system development.

6.2. Future Work

The following points can be added to our project in future:

- The program that we designed can generate skeleton code in (VB.NET). in future we can improve this program to generate skeleton programming codes in other languages like java, C++,C# ...etc.

- Drawing the static class diagram to be compatible with Microsoft Visio or any other drawing program.

6.3. Summary

The application works in effective way, and it is able to find static class diagram from software requirements as XML format and generate skeleton programming codes for the obtained classes in VB.NET programming language.

References

[1] Arman, N. and Daghameen K., " A Systematic Approach for Constructing Static Class Diagrams from Software Requirements " , International Arab Conference on Information Technology (ACIT2007) ,November 26-28 2007, Amman, Jordan.

[2] http://en.wikipedia.org/wiki/Class_diagram , Accessed Dec. 23, 2008 .

[3] http://searchcio-midmarket.techtarget.com/home/0,289692,sid183,00.html , Accessed Nov. 22, 2008 .

[4] http:// searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci810206,00.html , Accessed Nov. 23, 2008 .

[5] http://www.oop.csmartkid.com/#Before%20you%20start , Accessed Dec. 19, 2008 .

[6] http://www.smartdraw.com/tutorials/software/oose/tutorial_01.htm , Accessed Nov. 16, 2008 .

[7] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen: "Object-Oriented Modeling and Design", Prentice Hall 1990, ISBN 0-13-629841-9

[8] Wnekoop, J., and Russo, N., "Studying System Development Methodologies: An Examination of Research Methods," Information Systems Journal, Vol. 7, pp. 47-65, 1997.

[9] www.cfoster.net/articles/xmldb-business-case/. Foster, Charles (2008). "XML Databases and XML information exchange"., Accessed May. 16, 2009 .

[10] www.simonstl.com/articles/whyxml.htm , St.Laurent, Simon (1998). "Why XML?". Accessed May. 16, 2009 .

Appendix

See the attached CD.