COMPUTING RIPPLE EFFECT FOR WEB APPLICATIONS

by

NABIL BABA

B.S., Computer Science, Lebanese American University, 2002

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Division of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

Fall 2007
Thesis approval Form

Student Name: Nabil Baba I.D.: 19980585


Program: M.S. in Computer Science

Division/Dept: Computer Science and Mathematics

School: Arts and Sciences - Beirut

Approved/Signed by:

Nasha’at Mansour, Ph.D. (Advisor)
Professor of Computer Science

Mohamad Ladan, Ph.D.
Associate Professor of Computer Science

Sanaa Sharafeddine, Ph.D.
Assistant Professor of Computer Science

Date: Feb. 8, 2007
Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Nabil Baba

Signature: [Redacted] Date: 13/2/2007
I grant to the LEBANESE AMERICAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.
To my parents and beloved fiancée.
Acknowledgment

I would like to thank my advisor Dr. Nasha’at Masnour for his guidance throughout my M.S studies. A thanks is also to Dr. Ladan and Dr. Sharafeddine for being on my thesis committee.

I would like to express my sincere gratitude to the Lebanese American University whose financial support during my graduate studies made it all possible.

Finally, I would like to thank my family, fiancée and friends for their long and continuous help and support.
Abstract

The number of internet web applications is rapidly increasing in a variety of fields and not much work has been done for ensuring their quality, especially after modification. Modifying any part of a web application may affect other parts. If the stability of a web application is poor, then the impact of modification will be costly in terms of maintenance and testing. Ripple effect is a measure of the structural stability of source code upon changing a part of the code. It provides an assessment of how much a local modification in the web application may affect other parts. No previous work has been published on computing the ripple effect for web application. In this thesis, we propose, for the first time, a technique for computing ripple effect in web applications. This technique is based on direct-change impact analysis and dependence analysis for web applications developed in the .Net environment. Dependences include control dependences, data dependences, call dependences, semantic dependence, event-based dependences, functional dependences, and class dependences. Also, a complexity metric is proposed to be included in computing the ripple effect. A case study is used to show the applicability of our technique.
Contents

1. Introduction

2. Background on .Net Framework Infrastructure focusing on ASP.Net and VB.Net
   2.1 Introduction
   2.2 The .Net Framework
      2.2.1 Features of the Common Language Runtime
      2.2.2 Class Libraries
   2.3 ASP.NET architecture
      2.3.1 Limitations in Classic ASP
      2.3.2 Simple Page Inheritance
   2.4 Scope and Accessibility in ASP.Net and VB.Net
      2.4.1 Block scope
      2.4.2 Interface in VB.Net
      2.4.3 Code Behind in ASP.net
   2.5 Different ways in creating a .net website
      2.5.1 ASP.net pages structure type
      2.5.2 The Global .asax Application File
   2.6 ASP.NET components and detailed structure
      2.6.1 Web User Controls
      2.6.2 Server Controls

3. Dependences and Change Impact analysis in .Net Websites
   3.1 Dependences and Change Impact analysis in ASP.Net
      3.1.1 Dependences in ASP.Net
         3.1.1.1 Control Dependences
         3.1.1.2 Data Dependences
         3.1.1.3 Call Dependences
         3.1.1.4 Semantic Dependences
         3.1.1.5 Event-based Dependences
      3.1.2 Change impact analysis in ASP.Net
      3.1.3 Types of Changes and their relationship
   3.2 Dependences and Change Impact Analysis in VB.Net

Page
1
6
6
6
8
9
9
10
12
12
13
14
16
17
17
19
20
20
20
26
26
27
27
28
28
31
31
33
36
37
List of Figures

Fig 2.2: .Net Framework
Fig 2.3.1 (a): an example of an old ASP page
Fig 2.3.1 (b): Web Forms allow WYSIWYG design
Fig 2.3.1 (c): showing the ASP.net code resulted from a single button “Click me”
Fig 2.3.2: showing a simple page inheritance hierarchy
Fig 2.4.2: Interface code example in VB.Net and its Output
Fig 2.5.1 (a): Two pages created in ASP.Net each made from ASP/HTML and VB.Net code behind
Fig 2.5.1 (b): Two pages created in ASP.Net each made from ASP/HTML
VB.Net code behind and a script written in .aspx file
Fig 2.5.1 (c): Two pages created in ASP.Net each made from ASP/HTML and script written in VB without code behind
Fig 2.6.2 (a): Different kinds of controls
Fig 2.6.2 (b): an example of a button server control.
Fig 2.6.2 (c): A RangeValidator Example
Fig 3.1.2: .aspx page showing different types of containment
Fig 4.1: Login page created in ASP.Net made by .aspx and code behind
Fig 4.2: Code written in ASP/HTML for indexpage.aspx presentation file
Fig 4.3: Code written in VB.Net for code behind indexpage.aspx.vb
Fig 4.5: A three paged website each developed in a different way
Fig 5: Three pages that forms a website called Testlogin
Fig 5.1 (a): a user did not enter any value in the Login and Password in Indexpage
Fig 5.1 (b): The exact code written in ASP/HTML for indexpage.aspx presentation file
Fig 5.1.2: Code written in VB.Net for code behind indexpage.aspx.vb
Fig 5.2: a user chose to check the Fall 2006 semester schedule, it is displayed in a table
Fig 5.2.1: Code written in ASP/HTML for personal.aspx presentation file
Fig 5.2.2: Code written in VB.Net for code behind personal.aspx.vb
Fig 5.3 (a): Code written in VB.Net for the script in grades.aspx.vb
Fig 5.3 (b): a user chose to check the Fall 2006 semester grades, it is displayed in a table
List of Tables

Table 2.4: Accessibility Keywords in VB.Net  
Table 4.1: Running Time of all the steps in computing the Ripple Effect  
Table 5.5: testlogin Ripple Effect and Logical Stability results
Chapter 1

Introduction

The technological advancements during the last decade has changed the way individuals and businesses conduct business. Consumers and businesses are heavily relying on such technologies, mostly on the internet in order to perform their daily transactions. Two major standards have evolved for building enterprise web applications namely Microsoft ASP.NET and the Microsoft .NET Framework, and Apache's Struts along with the J2EE framework. In this research we will concentrate mainly on the Microsoft .Net and specifically on ASP.Net and VB.net.

But, behind the fascinating and entertaining frontal of a web site, lies the challenge in managing its infrastructure and its content. Web systems has a lot of challenges one of which compared to traditional software systems is the necessity and rate of change. This leads to the importance of managing these changes and controlling the change propagation.

Web applications and windows based software differ from each other in several ways, yet they have numerous similarities. They differ in the involvement of complex multi-tiered, heterogeneous architecture including web applications and database servers. Web applications also have its uniqueness. The boundaries of the web application in web platforms are the boundaries of the World Wide Web itself. Unlike normal closed software environment in windows applications, web applications are open to much more challenges. Understanding these challenges leads us to control the change propagation in the web environment.
Ripple effect measure has been identified as important and necessary within several software maintenance models. Imagine a stone being thrown into a pond; it makes a sound as it enters the water and causes ripples to move outwards to the edge of the pond. Transferring this image into a source code is easy. The stone entering the water is now the hypothetical change to the source code of a program, the effect of the change ripples across the source code via data flow. It is more important nowadays to ask questions about a program as a part of software development or maintenance such as: how much ripple is there? Which parts of the program affect parts the most?

The ripple effect reflects how possible it is that a change to a particular form, method or class is going to cause harms to the rest of a program or a web application. It helps in determining the scope of the change and presents a measure of the program complexity. It can also be helpful as being an indicator of the complexity and stabilities of a particular form, method, class or a program. Ripple effect was one of the earliest metrics concerned with the structure of a system and how its modules interact [Shepperd, M. 1993].

The internet is quietly becoming the body of the business world, with web applications as the brains. Websites are becoming entirely new in the world of software quality. This means that software faults in web applications have potentially disastrous consequences. Within minute in going online, a web application can crash due to the number of users requesting it or having a weak and unstable structure. The technical complexities of a website and variances in the web browser make testing and quality control more difficult. Computing the ripple effect and logical stability of a website would therefore help maintainers to achieve a more stable and reliable web applications. Ripple effect can show the maintainer what the effect of a change will be on the rest of the web application. It can highlight the fragile parts of the web application to be restructured for better stable performance.
A method for computing ripple effect was developed early by Yau and Collofello [Yau, S., Collofello, J. and McGregor, T. 1978] and enhanced over several years. It was proved difficult to write simple software using this algorithm; ripple effect tools have either taken an excessive amount of time to produce ripple effect measures or needed some user intervention to make critical decisions about the source code. Previous tools developed to produce ripple effect measures for procedural software have used Yau and Collofello's algorithm which is based on set theory. Several attempts have been made at using the algorithm to construct a tool to produce fast and accurate ripple effect measurements, none of which have completely succeeded. Black [Black, S. 2001] had reformulated the ripple effect algorithm using matrix arithmetic and used this reformulated algorithm to produce a software tool: REST [Black, S. 2001]. However, Black's algorithm computes ripple effect only for procedural programs.

[Salem, H. 2004] proposed ripple effect and logical stability measures for object oriented program using matrix arithmetic. They provided an analysis of object-oriented dependencies, relations, and propagations inside and outside classes. They also studied object-oriented complexity metrics and their relation to ripple effect and classify them as Intra-class metrics and Inter-class after introducing new object-oriented metrics. They calculated the ripple effect for object-oriented programs at the code level. Both Inter-class propagation and Intra-class propagation for each class were considered. They also calculated the architectural ripple effect at the system level. They developed an algorithm that clarifies the process of computing ripple effect.

and interactions among objects (include, submit and split). The proposed technique guarantees that all paths in the site which satisfy a selected criterion are properly exercised before delivery. And that helped us in understanding the way web applications are constructed and the flow between its objects.

[Ricca, F. and Tornella, P. 2002] also investigated in web application slicing and data flow testing of web applications. To apply slicing to web applications, the authors had to define web application’s specific dependences and build upon these dependences the corresponding system dependence graph (SDG) from which valuable slices can be extracted.

[Houri, M. 2004] presented new techniques for testing web applications in the .Net environment. They extended previous work on modeling web applications by enhancing previous dependence graphs and proposing an event-based dependence graph model. They applied a known data flow testing methods to the dependence graphs and proposed an event-flow testing method.

In this thesis we propose a technique to compute the ripple effect and logical stability for web applications focusing on .Net environment (ASP.Net and VB.Net) using matrix arithmetic. In order to compute the ripple effect of a .Net web application we provide an analysis of ASP.Net and VB.Net object-oriented dependencies, relations, and propagations locally and globally for ASP.Net (.aspx pages) and inside and outside classes for VB.Net and VBScripts. Also, we propose a complexity metric for the first time regarding ASP.Net code and it is applicable for VB.Net too to be included in computing the ripple effect. We compute the ripple effect for ASP.Net and VB.Net object-oriented at the code level. Both global and Form-Scope Propagation for .aspx pages (ASP.Net) and Inter-class propagation and Intra-class propagation for each class are considered. We also compute the architectural ripple effect at the system level. Each
matrix used within the algorithm holds a particular type of information about the software under study.

This thesis is organized as follows:

**Chapter 2** review the literature about .Net Framework Infrastructure focusing on ASP.Net and VB.Net. We take a deep look at ASP.NET and VB.Net architecture talking about the Scope and Accessibility and all the objects and controls created by ASP.Net and VB.Net. We analyze the different ways in creating and building a .Net web application.

**Chapter 3** focuses on the Dependences and Change Impact analysis in .Net Websites, giving in details all kinds of dependences that an ASP.Net and VB.Net embodies, and giving the types of changes and their relationship covering the impact analysis for each of ASP.Net and VB.Net languages. Stating in details the types of Change Propagations and their relationship, classifying them into Global and Form-Scope Change Propagation for ASP.Net and Intra-class and Inter-class change propagation for the VB.Net OOP.

**Chapter 4** gives a definition of the computation of ripple effect in .Net environment. It is divided mainly into two sections, one covering ASP.Net (.aspx pages) and the other is VB.Net (.aspx.vb). This chapter gives details of how and what to compute using the matrices.

**Chapter 5** describes in details how to apply our proposed approach on a .Net website example made from three different pages. It shows in step by step the computation of ripple effect and logical stability using matrix arithmetic.

**Chapter 6** analyzes the results obtained from the example in chapter 5 and concludes our approach. It also suggests a list of further work that will be helpful for the software development community.
Chapter 2

Background on .Net Framework Infrastructure focusing on ASP.Net and VB.Net

2.1 Introduction

Two major standards have evolved for building enterprise web applications namely Microsoft ASP.NET and the Microsoft .NET Framework, and Apache's Struts along with the J2EE framework. In order to compute the ripple effect of a web application written in .Net language (ASP.Net and VB.Net) it is a must to have a clear idea about .Net Framework Infrastructure and going into the details of ASP.Net and VB.Net. Note that VB.Net is pure Object Oriented Programming Language, but the way it interacts with ASP.Net and the additional abilities given to it makes all the difference compared to traditional OOP and that what makes its uniqueness.

2.2 The .Net Framework

Microsoft chose to pursue a different course of action than the one they used to follow, they decided to re-engineer their technologies rather than regroup them into different names. This radical change resulted in the emergence of the .Net Framework. The new framework concentrated on the needs of different enterprises from performance to security, reliability and availability. It was built to support, build and run the next generation of web applications and XML web services, and its main objectives as listed in [Microsoft Co. MSDN 2006] are:

- Providing a consistent object-oriented programming
- Providing a code-execution environment that minimizes software deployment and versioning conflicts.
- Providing a code-execution environment that promotes safe execution of code.
- Providing a code-execution environment that eliminates the performance problems of scripts.
- Making development consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- Ensuring that code based on the .NET Framework can integrate with any other code.

The two major components of the .NET framework are the common language runtime and the .NET Framework class library. The foundation of the framework is the common language runtime which acts like an agent that manages code at execution time [Microsoft Co. MSDN 2006]. It provides various services like thread management, memory management, and remoting. To understand the concept of code management, we should first differentiate between managed and unmanaged code. In [Microsoft Co. MSDN 2006] they defined it as follows: “Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code”.

The other main component of the .NET Framework is the class library which is “a comprehensive object oriented collection of reusable types that enables you to develop various types of applications (command line, GUI, ASP.NET…)” [Microsoft Co. MSDN 2006].

The unmanaged components host the .NET Framework and “load the common language runtime into their processes and initiate the execution of managed code, creating a software environment that can exploit both managed and unmanaged features” [Microsoft Co. MSDN 2006].

An example of an unmanaged application that hosts the runtime is Internet Explorer. Hosting the runtime using the internet explorer enables you to embed managed components or windows forms controls in HTML documents, making managed mobile code possible along with important improvements that only managed code can offer.
A relationship between the common language runtime and the class library to the applications programmed and to the overall system is illustrated in Fig 2.2. It shows the way managed code operates within a larger architecture.

![Fig 2.2: .Net Framework](image)

### 2.2.1 Features of the Common Language Runtime

The common language runtime manages thread execution, code execution, memory, code safety verification, compilation, and other system services [Microsoft Co. MSDN 2006]. These features are essential to the managed code that runs on the common language runtime.

The managed components are granted various degrees of authentication based on a number of factors that include their origin. In [Microsoft Co. MSDN 2006] they mentioned that the managed component may or may not be able to perform file access operations, registry access operations or other functions even if it is being used in the
same application. The security options of the runtime enables safe web applications to be run without compromising the personal data or file system security.

Moreover, the common type system (CTS) ensures that all managed code is self-describing. This means that Microsoft and other third party compilers generate managed codes that conform to the CTS.

This kind of infrastructure enables the developer to use managed code to write his own logic, while still enjoying the excellent performance of the industry's best enterprise servers that support runtime hosting.

2.2.2 Class Libraries

The .NET Framework provides a rich and extensive base class library. Classes are organized by namespaces depending on its functionality. For example, the user interfaces for the Web can be found in the System.Web.UI. The System.Web.UI.Page class contains the methods and properties needed for an ASP.NET page. The .NET Framework (1.1) base class libraries contains over 200 core (System.*) namespaces. More are added in .Net Framework (2.0)

2.3 ASP.NET architecture

In traditional ASP programming, as [Provost, P. 2002] pointed out that the page is divided into logical sections representing the header, left navigation, body, and footer elements. A separate include (for the files needed) is created for each of the common elements and the body section is placed in the actual ASP page. The page then includes the appropriate files to build up the page. This is a significant improvement, but still creates a few maintenance problems.

In ASP.Net developing a website is much easier and more secure. In the coming sections we will discuss the main features of ASP.Net and make some comparison with the old fashioned ASP way.
2.3.1 Limitations in Classic ASP

ASP.NET tags are different from normal ASP tags, they have a special format. They always begin with the prefix "asp:" followed by the class name or the tag used. If there is no closing tag, the tag must end with "/". Any attributes in the tag corresponds to a control property, aside from the runat="server" attribute, which declares that the control will be processed on the server. The web controls starts by inheriting from the WebControl base class.

```html
<html>
<body bgcolor="yellow">
<center>
<h2>Hello World!</h2>
<p>%Response.Write(now())%</p>
</center>
</body>
</html>
```

Fig 2.3.1 (a) : an example of an old ASP page

The code in Fig 2.3.1 (a) illustrates a limitation in Classic ASP: The code block has to be placed in a place where the developer wants the output to appear. [W3Schools, 2005] stated that with Classic ASP it is unattainable to separate executable code from the HTML itself. This makes the page difficult to read and maintain.

With the introduction of ASP.NET, developers have been giving a powerful new set of tools to help resolve these problems. ASP.NET uses an object-oriented development paradigm. Classes in ASP.Net provides a number of services to the web developer including caching, rendering, response and request access, etc [Provost, P. 2002].

In contrast to classic ASP, none of the code in an ASP.NET page is interpreted; it is compiled into an assembly just like any other .NET library the first time it is requested.
1. First, the compilers check code syntax, method parameters and return values.
2. Second, ASP.NET pages execute faster than classic ASP pages because compiled code is more organized than script that is interpreted for every request [TOPXML, 2006].

On the server-side, the web controls look like HTML, where the designer of the web forms can create for us, an example is illustrated in Fig 2.3.1 (b)

![Web Forms design](image)

Fig 2.3.1 (b) Web Forms allow WYSIWYG design

The ASP.NET code for the page looks somewhat like this:

```html
<html>
<body>
<form runat="server">
   
   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 150px; POSITION: absolute; TOP: 130px"
   runat="server" Text="Click Me">

   </asp:Button>

</form>
</body>
</html>
```
2.3.2 Simple Page Inheritance

Object-oriented programming is the new look of ASP.Net. All ASPX pages derive from System.Web.UI.Page, we will be able to create a base class that stands between our page class the global root Page class as shown in Fig 2.3.2. This class is highly responsible in producing the template used in a website. By doing so, we will be able to make the page classes derive from the base class, containing only the markup and server controls needed for their specific task.

![Diagram showing page inheritance hierarchy]

2.4 Scope and Accessibility in ASP.Net and VB.Net

Generally, in a simple ASP.Net application most of the variables will be private because of the majority of the code that will be self-contained in a single web page class. When creating separate components to reuse functionality, however, accessibility becomes much more important.
Accessibility is defined on a class-by-class basis. The accessibility keywords are listed in table 2.4:

<table>
<thead>
<tr>
<th>VB.net keyword</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Can be accessed by any other class.</td>
</tr>
<tr>
<td>Private</td>
<td>Can only be accessed by code procedures inside the current class. This is default if you don’t specify an accessibility keyword.</td>
</tr>
<tr>
<td>Friend</td>
<td>Can be accessed by the code procedures in any of the classes in the current assembly.</td>
</tr>
<tr>
<td>Protected</td>
<td>Can be accessed by code procedures in the current class, or any class that inherits from this class.</td>
</tr>
<tr>
<td>Protected Friend</td>
<td>Can be accessed by code procedures in the current application, or any class that inherits from this class.</td>
</tr>
</tbody>
</table>

You can use initializers with class variables in the same way that you use them with procedure level variables. These variables will be automatically created and initialized when the class is first created.

2.4.1 Block scope

In traditional Visual basic, there were only two types of scope for a variable: form level and procedure level. VB.net tightened these rules by introducing block scope. Block scope means that any variable created inside a block structure (such as conditional IF...End if or a For... Next or a Do... Loop) are only accessible inside that block of code.

For i = 0 To 10
    Dim sum as integer
    ' (Do some calculation with sum.)
Next
    ' you cannot access sum variable here.
This change will not affect many programs. It's really designed to catch a few more accidental errors.

2.4.2 Interface in VB.Net

Interfaces are used when you have two or more classes that represent the same public members, but carry out different purposes. It is used to create definitions for component interaction. After working and implementing interfaces in VB.Net you notice that it is another way of implementing polymorphism. We can specify methods that a component must implement (method prototype) without specifying how the method is implemented. By creating an interface we specify the methods in it and its up to the class to decide how these methods are implemented. Multiple inheritance are not supported directly by VB.Net but it can be achieved by using interfaces. We use the Interface keyword to create an interface and implements keyword to implement the interface. Once an interface is created the implementation off all the methods specified in that interface is required.

The following code in Fig 2.4.2 shows an example that demonstrates the use of interface, taken from (http://www.startvbdotnet.com/oop/interface.aspx):

```vbnet
Imports System.Console
Module Module1

Sub Main()
Dim OneObj As New One()
Dim TwoObj As New Two()
'creating objects of class One and Two
OneObj.disp()
OneObj.multiply()
TwoObj.disp()
TwoObj.multiply()
'accessing the methods from classes as specified in the interface
End Sub

End Module

Public Interface Test
'creating an Interface named Test
Sub disp()
Function Multiply() As Double
'specifying two methods in an interface
```
End Interface
'notice that the implementation of the disp and the multiply
'methods are not specified

Public Class One
  Implements Test
  'implementing interface in class One

  Public i As Double = 12
  Public j As Double = 12.17

  Sub disp() Implements Test.disp
    'implementing the method specified in interface
    WriteLine("sum of i+j is" & i + j)
    Read()
  End Sub

  Public Function multiply() As Double Implements Test.Multiply
    'implementing the method specified in interface
    WriteLine(i * j)
    Read()
  End Function

End Class

Public Class Two
  Implements Test
  'implementing the interface in class Two

  Public a As Double = 20
  Public b As Double = 32.17

  Sub disp() Implements Test.disp
    WriteLine("Welcome to Interfaces")
    Read()
  End Sub

  Public Function multiply() As Double Implements Test.Multiply
    WriteLine(a * b)
    Read()
  End Function

End Class
Output of above code is the image below.

![Image of code output]

Fig 2.4.2: Interface code example in VB.Net and its Output

2.4.3 Code Behind in ASP.net

ASP.NET offers a new option where we can place code in a file that is called a "code-behind file". This file can hold the definition for the base class of the actual page. The page can access methods and properties from that base class which makes the code-behind file a good place for the methods we call from the page template [TOPXML, 2006]. In this way, the mess will be out of the HTML templates, but more significantly, the code-behind page is built into a regular .NET assembly as part of the regular build process of a web application [TOPXML, 2006]. It is compiled before the deployment of a web application to let the compiler detect coding errors in your pages.

Code behind represents a major change in philosophy that brings ASP.Net in line with modern programming practices. Code behind includes:

- **Better organization:**
  ASP pages are infamous of their tangle of the script commands and HTML formatting tags. This confusion often results in a spaghetti-like mess that is difficult to debug, impossible to reuse, and even harder for other programmers to work with. With code-behind, the code is encapsulated in a neat class. It's easily to
read and easier to isolate and reuse pieces of useful functionality. The only way you could do this with ASP was to create COM components.

- **Separation of the user interface:**
  One of the most well-known problems with traditional ASP development is that it mingles formatting and content with programming logic. This means that programmers are often the only ones able to perform the graphic design for an ASP website. Else you need to give the source code to a professional web designers raising the possibility that sensitive information will leak out or manipulated. With code-behind, the .aspx file can be manipulated, polished and perfected by any other user as long as the control names remain the same.

- **The ability to use advanced code editors:**
  Embracing code-behind programming allows you to benefit from essential tools—VS.Net can automatically verify the syntax of the code-behind files, provide IntelliSense statement completion, and allows to design the corresponding .aspx webform.

### 2.5 Different ways in creating a .net website

After practicing .net development for creating asp.net website, I have found that in order to calculate the ripple effect of a .net website we have to consider the different ways in creating it first. Because every way has its own way in calculating its ripple effect, especially that a website is not made from one page only it can be hundreds of pages all written in different ways. Section 2.5.1 will spot the light on the different ways to create a website and according to that we will show how to calculate the Ripple effect of each case.

#### 2.5.1 ASP.net pages structure type

Mainly we will discuss three cases/types in creating an ASP.net page. All can result the same functionality but they are all different in structure. We can create a .net page in the following three ways:
a. Clear ASP/HTML code in .aspx representation file and VB.NET code in .aspx.vb (code behind)

Fig 2.5.1 (a): Two pages created in ASP.Net each made from ASP/HTML and VB.Net code behind

b. Heterogeneous code (ASP/HTML and VB as a script) in .aspx file and VB.Net code in asxp.vb (code behind)

Fig 2.5.1 (b): Two pages created in ASP.Net each made from ASP/HTML VB.Net code behind and a script written in .aspx file

c. Only one .Net page which is .aspx page that contains the whole VB.Net code found in .aspx.vb (code behind) written as a script
The reason behind defining the different types and cases in creating a .net page is because the website can be written by different programmers, and every programmer has a different style in creating a page, some rely only on code behind file (first case in Fig 2.5.1(a)) with the .aspx file, others work with scripts in .aspx file and the code behind too (second case in Fig 2.5.1(b)) and at last some prefer not to write any code behind file so they write it in a script (third case in Fig 2.5.1(c)). Although the third case is not advised by any experienced programmer that is because the designer who wants to add colors and animation to the website will deal with the .aspx file, so he might delete a method for an example that is essential for the functionality of the website by mistake and will not know how to write it again.

2.5.2 The Global .asax Application File

The global .asax file allows us to write global application code. The global .asax file looks similar to a normal .aspx file, except for the act that it cannot contain HTML or ASP.NET tags. Instead, it contains event-handling code that reacts to application or session events. Each ASP.NET application can have one global.asax file. Once it is placed in the appropriate virtual directory, ASP.NET will use it and recognize it automatically. It also supports code-behind development.
2.6 ASP.NET components and detailed structure

ASP.Net contains a rich library in components of different kinds and uses, in this section we will discover all the important components that are essential to our research.

2.6.1 Web User Controls

When getting started with ASP.NET a developer will first notice the new style of control that has been introduced: “Web User Controls”. It allows a developer to encapsulate a common chunk of HTML/server-side code into a component that can be reused on many different pages.

They are not used in a page using the #include directive. Instead they are either placed in the ASPX file as a custom tag or from server-side code with the LoadControl statement.

2.6.2 Server Controls

Server controls are tags that are understood by the server. ASP.Net turned the HTML tags into controls that can be programmed on the server side.

There are three kinds of server controls:

1. HTML Server Controls : Traditional HTML tags
2. Web Server Controls : New ASP.NET tags
3. Validation Server Controls : For input validation

- **HTML server controls** are server based equivalent for standard HTML elements. These controls are ideal and useful in migrating existing ASP pages into ASP.NET. HTML server controls provides three key feature:

  1. They generate their own interface: you can simply set the properties code, the underlying HTML tag is updated automatically when the page is rendered and sent to the client.
2. **They retain their state**: there is no need to recreate a web page from scratch each time you send it to the user.

3. **They fire events**: in ASP code, everything was grouped into one block that executes from start to finish. With the new approach of event-based programming, responding to the individual user actions and creating more structured code became easier.

HTML controls are a compromise between web controls and traditional Asp.net programming. They use the familiar HTML elements but provide a limited object-oriented interface. Essentially HTML controls are designed to be straightforward, predictable, and automatically compatible with existing programs. With Html controls the final HTML page that is sent to the client closely resembles the original .aspx page.

When the special attribute runat="server" is added to a tag, that means that tag is transformed into a server control. Note that when the id attribute is added, it interacts with in the code; it assigns a unique name that will be used to refer to the control in the code.

Every HTML tag in .NET is inherited from the HtmlControl base class this special link between Object oriented programming and normal tag writing is essential where you can still use the old way in creating web pages and the advanced web programming structure. In Fig 2.6.2 (a) below you will find a list of all HTML Tags, the class it represents and the event it triggers:
<table>
<thead>
<tr>
<th>HTML Tag</th>
<th>Class represented</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;a&gt;</code></td>
<td>HtmlAnchor</td>
<td>ServerClick</td>
</tr>
<tr>
<td><code>&lt;button&gt;</code></td>
<td>HtmlButton</td>
<td>ServerClick</td>
</tr>
<tr>
<td><code>&lt;form&gt;</code></td>
<td>HtmlForm</td>
<td>ServerClick</td>
</tr>
<tr>
<td><code>&lt;img&gt;</code></td>
<td>HtmlImage</td>
<td></td>
</tr>
<tr>
<td><code>&lt;input type=&quot;button&quot;</code>, <code>&lt;input type=&quot;submit&quot;</code>, and <code>&lt;input type=&quot;reset&quot;</code></td>
<td>HtmlInputButton</td>
<td>ServerClick</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;checkbox&quot;</code></td>
<td>HtmlInputCheckBox</td>
<td>ServerChange</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;text&quot;</code>, <code>&lt;input type=&quot;submit&quot;</code>, and <code>&lt;input type=&quot;file&quot;</code></td>
<td>HtmlInputControl</td>
<td></td>
</tr>
<tr>
<td><code>&lt;input type=&quot;file&quot;</code></td>
<td>HtmlInputFile</td>
<td></td>
</tr>
<tr>
<td><code>&lt;input type=&quot;hidden&quot;</code></td>
<td>HtmlInputHidden</td>
<td>ServerChange</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;image&quot;</code></td>
<td>HtmlInputImage</td>
<td>ServerClick</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;radio&quot;</code></td>
<td>HtmlInputRadioButton</td>
<td>ServerChange</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;text&quot;</code> and <code>&lt;input type=&quot;password&quot;</code></td>
<td>HtmlInputText</td>
<td>ServerChange</td>
</tr>
<tr>
<td><code>&lt;select&gt;</code></td>
<td>HtmlSelect</td>
<td>ServerChange</td>
</tr>
<tr>
<td><code>&lt;table&gt;, &lt;tr&gt;, &lt;th&gt; and &lt;td&gt;</code></td>
<td>HtmlTable, HtmlTableRow, HtmlTableCell</td>
<td></td>
</tr>
<tr>
<td><code>&lt;textarea&gt;</code></td>
<td>HtmlTextArea</td>
<td>ServerChange</td>
</tr>
<tr>
<td>Any other HTML element ex: <code>&lt;b&gt;, &lt;i&gt;, &lt;u&gt;, &lt;font ... &gt;</code></td>
<td>HtmlGenericControl</td>
<td></td>
</tr>
</tbody>
</table>

Fig 2.6.2 (a): Different kinds of controls

A **ServerClick** is simply a click that is processed on the server side, it is provided by most button controls and it allows the code to take immediate action. The **ServerChange** event responds when a change has been made to a text or selection control.
Note that any HTML control that requires a closing tag also inherits from the HtmlContainer control. The properties of the tags can be set either in the code (using VB.net or any other .Net language) on in the tag itself.

**Note:** All HTML server controls must be within a `<form>` tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.

- **WEB server controls** are similar to the HTML server controls, but provide a richer object model with a variety of properties for style and formatting details, more events, and a closer parallel Windows development. They are special ASP.NET tags that are understood by the server. Web controls feature some user interface elements that have no HTML equivalent such as DataGrid and validation control. Additional web controls are needed because HTML controls are still more limited than server controls need to be. Every HTML control corresponds directly to an HTML tag, meaning that the programmer will be bounded by the limitations and abilities of the HTML language. Web controls, on the other hand, emphasize the future of web design:
  - **They provide rich user interface:** the control creates the required HTML tags for the programmer.
  - **They provide a consistent object model:** the HTML language is full of spaghetti-like code especially if the control added needs to be filled with different properties. With web controls depending on the properties the programmer sets, the HTML element will be created automatically.
  - **They tailor their output automatically:** the programmer don’t need to know about the client side and what type of browser he/she is using. ASP.Net server controls can detect the type of browser, and automatically adjust the HTML code written to take advantage of its features such as support for DHTML.
  - **They provide high-level features:** web controls allows the programmer to access additional events, properties, and methods that don’t correspond directly to a typical HTML controls.
In the following example (Fig 2.6.2 (b) ) we declare a Button server control in an .aspx file. Then we create an event handler for the Click event which changes the text on the button:

```html
<asp:Button id="button1" Text="Click!" runat="server" OnClick="submit"/>
```

Fig 2.6.2 (b): an example of a button server control.

- **Validation server controls:** is used to validate user-input. If the user-input does not pass validation, it will display a message to the user indicating an error. Each validation control performs a specific type of validation (like validating against a specific value or a range of values) [W3Schools, 2005].

Page validation is performed when a ImageButton, Button, or LinkButton control is clicked by default. A validation can be prevented when a button control is clicked by simply setting the “CausesValidation” property to false. The syntax for creating a Validation server control is:

```html
<asp:control_name id="some_id" runat="server"/>
```
In the following example (Fig 2.6.2 (e)) we declare one TextBox control, one Button control, and one RangeValidator control in a .aspx file. If validation fails, the text "The value must be from 1 to 100!" will be displayed in the RangeValidator control:

```html
<html><body>
<form runat="server">
Enter a number from 1 to 100:
<asp:TextBox id="tbox1" runat="server" />
<br /><br />
<asp:Button Text="Submit" runat="server" />
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
EnableClientScript="false"
Text="The value must be from 1 to 100!"
runat="server" />
</form></body></html>
```

Fig 2.6.2 (e): A RangeValidator Example
Chapter 3

Dependences and Change Impact analysis in .Net Websites

As we have noticed, .Net framework provided us a rich and easy way in creating a webpage. To recall, a .Net webpage is mainly divided into two main files: a file that contains the ASP.Net code (.aspx) and a file that implements the controls and events that are found in ASP.Net page using pure object oriented programming language called VB.Net (.aspx.vb). That is why if we want to study the Change Impact analysis and dependences in web pages using .Net framework, we have to study the change impact analysis and dependences in both ASP.Net and VB.Net. Some work related to web application slicing in (Ricca and Tonnella, 2001) and (Ricca and Tonnella, 2002) contributed to webpage dependences, they will be discussed clearly and additional dependences are added to cover the ASP.Net architecture. Previous work done by (Mansour and Salem, 2004) covered all the dependences found in Object Oriented architecture. Although VB.Net is a pure Object Oriented Language but it contains some additional dependences that will be discussed in section 3.2 in this chapter.

3.1 Dependences and Change Impact analysis in ASP.Net

In order to understand the change propagation in any type of application, we should come up with its dependences and the impact of change. ASP.NET consists of entities and controls that functions in different ways. These entities and controls relate to each other in different ways therefore creating the propagation of a change obtained in an .aspx file. Moreover ASP.Net can contain scripts instead of code behind, the dependences of the scripted code is the same as VB.Net dependences so in this section we will discuss
the general dependence type that the VB script code could have and more details will be followed up in section 3.2.

3.1.1 Dependences in ASP.Net

ASP.NET consists of entities and controls that functions in different ways. These entities and controls relate to each other in different ways therefore creating the propagation of a change obtained in an .aspx file. In this section we will discuss in details the different types of dependences that can be found in ASP.Net web applications.

3.1.1.1 Control Dependences

Web applications are characterized by a kind of control dependence found in the HTML/ASP code. In general, for the server/client sided code; control dependences connect the predicate tested at a conditional or loop statement to the instructions the execution of which directly depends on the true value of the predicate. HTML/ASP statements can not be interpreted correctly unless all enclosed tags are available, that is because the browser itself acts as a compiler that compiles the code to ensure that the syntax to get executed is right. When opening a “TAG” you will be defining a new scope, control dependence holds between a tag and all directly enclosed statements.

Definition: a control dependence holds between two statements if the former defines a scope which directly includes the latter. [Ricca, F. and Tonnella, P. 2002]

A note worth mentioning here is that creating a “WebForm” makes a general scope that might include different types of controls and data connectors in the .aspx page. When defining an object in the form you cannot define another one having the same name “ID” in another “WebForm” in the same .aspx page. .Net applications are so closely coded like normal web pages in terms of HTML tags where they have to be enclosed properly. Therefore, come to think of a .aspx page as a connected graph we can imagine a statement inside a scope connected as a child to the current scope node (eg: a Form and a datagrid created inside it).
3.1.1.2 Data Dependences

Data dependences are useful to represent the information flaws in the application. They may reveal the presence of undesirable possibilities, such as using a variable not yet defined or using an incorrect definition of a variable.

An aspx page can be divided into two parts: the scripts that include all server side events and functions they can be written in any scripted language but we will focus on VBScript, and the HTML/ASP tags. As stated by [Ricca, F. and Tonnella, P. 2001], data dependences for normal traditional programs holds when a statement defines the value of a variable, which is used at another statement after being propagated to it along a definition clear path (i.e., a path containing no redefinition of the given variable). In a webpage data flaws may occur from server/client side statements to the HTML code. Moreover, a variable definition at an HTML statement can reach a server/client instruction only through a function call either directly to the function that is responsible to that variable -written as a script in the same page- or to the function that is found in a class as a method found in the .aspx.vb page, the variable can be sent as a submission parameter.

*Definition:* a data dependence holds between two statements if the former defines the value of a variable which is used by the latter, and a definition clear path exists between the two. [Ricca, F. and Tonnella, P. 2001]

3.1.1.3 Call Dependences

In a .Net web pages the calling dependences is totally different from normal procedural programs. Even the normal web pages that are made using different types of editors such as Microsoft Frontpage or Dreamweaver, differ in the calling dependences issue. That is because a .Net web page is made from two separate files yet they are so interconnected together, the .aspx and .aspx.vb files that makes the .Net webpage are different in structure. The .aspx.vb contains the code behind that the .aspx depends on,
and it is made in pure Object Oriented Structure where we will discuss its dependences in section 3.2. The .aspx file contains all the HTML and ASP code, but yet it can contain a lot of procedures that can be created using scripts (VBscripts). So once we want to check the call dependences in a .aspx page we have to take care of the normal HTML calls, the ASP calls to the code behind functions and to take care of the scripts call that will be found in this case in the client side.

According to [Ricca, F. and Tonnella, P. 2001], in web applications data flows can only be propagated inside an invoked server side script and never back from it. An invoked procedure/function can return some value to the calling page. The definition provided was

**Definition:** a call dependence holds between each statement of type Call and the server/client program or procedure invoked. A Parameter-in dependence holds between any actual parameter of a call and the respective formal parameter of the invoked program or procedure. [Ricca, F. and Tonnella, P. 2001]

But in fact .Net web application call dependences are varied because of its unique structure as we mentioned earlier. [Houri, M. 2004] mentioned that we can have two types of call dependences, we can have inheritance call dependence as well as internal call dependence. But because we are separating the dependences in .Net web applications, we will not discuss the internal call dependence in details because it has a strong relationship with OOP VB.net. The .aspx file contains another kind of Call Dependence which is Page Call dependence

- **Inheritance Call Dependence:** as we mentioned earlier .aspx files has another file that it depends on sometimes. This file is the code behind file, in order to include that file in the .aspx one, we should inherit it. Inheritance Call dependence occur only at the root level of the .aspx file
**Definition:** An inheritance call dependence holds between a code behind class .aspx.vb and a presentation file .aspx if the keyword “inherits” of the aspx file explicitly declares this inheritance.

- **Page Call Dependence:** there are two types of pages, one type that can be called or accessed from more than one page (ex: index page) and another one that can be accessed only from one page such as frames and pop up pages.

**Definition:** a Page Call Dependence holds between a page and another either by clicking a link or triggered automatically when a page is loaded.

- **Internal Call Dependence:** it might be present in two ways: either in the .aspx code page as a script of any language -basically we are working with the VB script- or an .aspx.vb code behind classes which was discussed earlier. We will discuss more dependences in section 3.2 discussing the Object Oriented dependences and the change impact. The definition will change a little to cover the ASP.Net environment.

**Definition:** An internal dependence holds between a calling statement in a code behind class or a script (VB or any other language) and a method, if both the calling statement and the method are internal to the class.

- **Cascading Call Dependence:** this kind of calling dependence gives the ASP.NET pages its uniqueness. The unique feature of code behind, it enforces this kind of dependency because of the following:
• Many of the main elements of a .net webpage are defined first on the presentation files (the .aspx file) and used in the code behind file (.aspx.vb).

• Or, these elements leads to the definition of other data where the latter is used in the code.

**Definition:** a cascading dependence holds either (i) between a data definition in the presentation file and its use in the code behind or (ii) between a data definition in the presentation file and its use in the definition of another data in the code behind class. [Houri, M. 2004]

### 3.1.1.4 Semantic Dependences

Semantic dependences were presented concisely by [Ricca, F. and Tonnella, P. 2001], note that this dependence is important in the .net environment because it pays attention to the graphical and textual interface of the web page without neglecting the informative part.

**Definition:** a semantic dependence holds between an informative object (graphical, textual, and processing) and a page or another informative object if the former provides information on the latter. [Houri, M. 2004]

### 3.1.1.5 Event-based Dependences

Additional types of dependences are needed to satisfy one of the most important main features of ASP.net which is “Events”. There are different types of events in ASP.net some are controlled using event handlers that can be found in the code behind .aspx.vb file and other can be found in the scripts included in the same presentation files. And some triggered directly while writing the controls. These dependences are the following:
• Link dependence

When clicking a button, a lot of events might occur; the user for example can be taken directly into another page, or a pop up window might occur. This was done using an event that will fetch the requested page.

**Definition:** a link dependence holds between two pages (or more) if the first requests the second through an event, either automatically (pop up window) or by pressing a button. [Houri, M. 2004]

• Visible Effect Dependence

When clicking on a button after choosing an item from a menu, or editing some text forms, an event will be triggered taking the user directly into another page viewing the effect that he chose. For example: editing an e-card by changing its color and background, you can preview it by clicking on the preview button that will lead you to another page that contains the edited e-card.

**Definition:** a visible effect dependence holds between two pages if the first modifies the second through an event that will (1) implement the modification and (2) show the effect on the desired page by taking the user directly to it. [Houri, M. 2004]

• Invisible Effect Dependence

Web pages can be implemented differently; some pages can work in a different way than the “Visible effect” web pages. Sometimes you choose the event that you want to webpage to handle such as changing a background color of an e-card. And when you do change it there is no button that can lead you to a page contains the changed e-card, in that case you have to go to
the page manually to check if the change occurred or not. This is called the invisible effect dependence.

**Definition:** Invisible effect dependence holds between two pages if the first modifies the second through an event that will (1) implement the modification and (2) will not show the result of the effect on the desired page by taking the user directly to it. The user must go there “manually” to see the effect. [Houri, M. 2004]

### 3.1.2 Change impact analysis in ASP.Net

ASP.Net differs from normal programming languages, it has its own uniqueness. Although it is the new generation of web programming and traditional ASP, it is still sharing some common features such as “Tags” and “Scripts”. ASP.Net now uses objects that calls methods and classes that are inherited from the code behind file or written in the script section. Asp.net consists of two main controls that trigger events in order to complete the functionality of a .net website (they were discussed earlier, but worth mentioning here):

1. **Web User Controls:** to recall; User Controls allows a developer to encapsulate a common chunk of HTML or server-side code into a component that can be reused on many different pages.

2. **Server Controls:**
   a. HTML Server Controls: Traditional HTML tags
   b. Web Server Controls: New ASP.NET tags
   c. Validation Server Controls: For input validation

Mainly we will look in our analysis at the tags scope which is composed of multiple lines of code containing homogeneous/heterogeneous types of controls.

Looking at the way tags can be written, we will notice that every control has its own scope. The “Form” usually has the widest scope because it can contain all the server controls and web control calls. We conclude a direct relationship as follows:

*There is a direct relationship R between an opening Tag T and control C (TRC) if T and C have one of the two kinds of relationships: Containment or Use.*

33
According to (L. Li, 1998) in the Direct relationship were defined as follows:

- **Containment**: Class A contains class B if B declared as a class member of A.[Li, L. 1998] Changing it to fit ASP.Net programming we refer to containment as: Tag T contains control C if C is defined inside the scope of T (before the closing tag of T).

Ex:

```html
<form id="Form1" method="post" runat="server">
  <div style="display: inline; z-index: 102; left: 140px; width: 330px; position: absolute; top: 9px; height: 50px; ms-positioning="FlowLayout">
    <p align="center">
      <font size="6"> <strong>Test</strong> </font>
      </p>
  </div>
</form>
```

Fig 3.1.2: .aspx page showing different types of containment
We notice in Fig 3.1.2 that we have different type of containment, not just a tag containing a Control, but a tag containing a format for a label in line 4. “Test” is contained in the U tag and STRONG tags and both are contained in Font Tag and Font Tag belongs to P tag and P tag belongs to the DIV tag and finally DIV tag belongs to FORM tag. But bare in mind that this kind of containment is not what we are looking for, because its for the graphic designer who wishes to give a nice look for the web page, what we are looking for is the functionality containment that hits directly the performance and the way the webpage will behave. So what we are looking for are the following containment relationships:

- “Check” in line 6 is contained in Form1 in line 1
- “Password” in line 7 is contained in Form1 in line 1
- “age” in line 8 is contained in Form1 in line 1
- Lines 9 – 15 are contained in dropdownlist asp tag in line 8 and indirectly contained in Form1

- **Use/Reference:** there are different ways that the Use/reference can be formed:
  
  - **Containment:** if item I is inside the scope of control C, then control C uses I. ex: line 9 to 15 are all inside the scope of the drop down list “age” in line 8. So “age” uses all items starting from line 9 to 15.
  - Control Values passed to methods (scripted or in code behind) as parameters: If control C1 uses value V of control C2 then C1 uses C2. Ex: the submit button sends values (text field values) to a method on the server side.

- **Inheritance relationship:** all of the .aspx file inherits the code behind class and other classes. The classes/objects in the code behind file are accessed through the controls written in .aspx presentation file.
3.1.3 Types of Changes and their relationship

Changes found in ASP.Net pages specifically in .aspx presentation file are different from normal web programming changes. They are limited by the changes in the tags because they are the major code dealt with in .aspx other than the scripting languages which is pure Object Oriented Programming language in VB.Net and its changes is discussed in section 3.2 for the ASP.Net changes in .aspx presentation file we have the following:

i) System level change:
   a. Add a form
   b. Delete a form
   c. Change inheritance/code behind source
   d. Add/Change/Delete User Control source

ii) Form level change
   a. Add Web Server Control
   b. Delete Web Server Control
   c. Add HTML Server Control
   d. Delete HTML Server Control
   e. Add Validation Server Control
   f. Delete Validation Server Control
   g. Add User Control
   h. Delete User Control
   i. Change Form Method
   j. Change Form Runat

iii) Control Level Change
   a. Change Control Name
   b. Change Control Event
   c. Change Control Type
      i. HTML control to HTML server control
      ii. Change Control Runat
We still have to mention the effects in the system level of VB scripts if added to a .aspx file, but because we are separating the OOP (VB.Net) changes from ASP.Net (ASP Tags/HTML Tags) changes, then the scripts' changes (VBScript) will be discussed in section 3.2 while discussing VB.Net change impact analysis and its dependences. Moreover, we have to mention the graphical changes that might occur in the form level, but because we are focusing only on the functionality of the web site we did not mention them because they do not have a ripple effect in the web site. The Text/Colors/Graphics are for the graphic designers to worry about, a website can work perfectly without any picture/color added to it, but it will not work properly if an important control was missing or any of the changes discussed before occurred.

Inheritance should have the highest impact power over the containment and use, due to its importance in ASP.Net programming. Although a user might find that inheritance is not used a lot in .aspx file, but inheritance statement is included in the header of .aspx file. And without it, none of the controls will function because the implementation of their code is written in the code behind file .aspx.vb. Containment has the second greater impact power over the use.

3.2 Dependences and Change Impact Analysis in VB.Net

Lately a huge shift in Microsoft Visual Studio's VB 6.0 programming language happened. Visual Basic 6.0 became more powerful, easier and well constructed. VB 6.0 changed into an Object Oriented programming language called VB.Net. VB.net is used with the ASP.Net webpage in two ways, either through the code behind file or by using the scripts in the .aspx presentation file. Visual Basic .NET is a fully object-oriented programming language, that means it supports four basic views of object-oriented programming: abstraction, encapsulation, inheritance, and polymorphism.

We have already noticed many of these object-oriented concepts by just looking at the objects that surround us in our everyday lives. The objects that will be constructed with VB.NET will live out their lives within the .NET Framework. They can be shared in library class the can be easily inherited whenever needed.
VB.NET can be used to create three different types of programs:

- Web applications
- Windows applications
- Console applications

The VB.NET language was developed to allow .NET programmers to create very large, powerful, high-quality web applications quickly and easily. The .NET technology for creating web applications is ASP.NET, that is why the most important languages to make a powerful web application are ASP.Net and VB.Net.

Previous work in Object Oriented dependences was done by [Salem, H. 2004] covered all the Object Oriented Dependences, but because VB.Net is a new language with a new style in structure two additional dependences will be covered more. In section 3.2.1 we will discuss the previous work in Object Oriented dependences and add up to them.

Understanding Object Oriented dependences and the change impact is important because both will provide us with a great idea about the error/change propagation in order to be able to calculate the ripple effect and decide how a change ripples in VB.Net. Object Oriented systems maintenance is difficult for several reasons: understanding the combined effect or combined functionality of the member function is extremely difficult. Object Oriented complex relationships between classes make it difficult to foresee and identify the ripple effect of changes. The data dependences, control dependences, and state behavior dependences make it difficult to initialize test cases and generate test data to efficiently retest the impacted components. Finally, complex relations also make it difficult to define a cost effective test strategy to retest the impacted components [Li, L. 1998].

3.2.1 VB.net Object Oriented Dependences

Wilde and Huitt, were one of the earliest researchers that introduced the term *dependency* in a software system. They said that *dependency is a direct relationship between classes*
**X** and 
**Y** entities in the system 
**X R Y** such that a modification to **X** may affect **Y**. [Wilde, N. and Huit, R. 1992]

They classified dependences as:

- Data dependences between two variables
- Calling dependences between two modules
- Functional dependences between a module and the variables it computes.
- Definitional dependences between a variable and its type.

[Salem, H. 2004] presented more details and classified them in five types of changes that may occur in object-oriented code, the five types of changes are adapted by VB.net and three additional will be listed below:

i. Class-to-Class Dependences

   a) C1 is a direct super class of C2 (C2 inherits from C1)
   b) C1 is a direct sub class of C2 (C1 inherits from C2)
   c) C1 is an ancestor class of C2 (C2 indirectly inherits from C1)
   d) C1 uses C2 (C1 references C2, include direct reference and indirect reference)
   e) C1 contains C2
      ▪ C1 contains C2 by Value
      ▪ C1 contains C2 by reference

ii. Class to Method Dependences

   a) Method M returns object of Class C
   b) C implements method M

iii. Class to Variable Dependences

   a) V is an instance of Class C
   b) V is a class variable of C
   c) V is an instance variable of C
d) \( V \) is defined by Class C

iv. Method to Variable Dependences

a) \( V \) is a parameter for method M
b) \( V \) is a local variable in method M
c) \( V \) is imported by M (non local variable used in M)
d) \( V \) is defined by M

v. Method to Method Dependences

a) Method M1 invokes method M2
b) Method M1 overrides M2

This research will introduce other types of change as follows:

vi. Variable to Block

To recall; Block scope means that any variable created inside a block structure (such as conditional IF...End if or a For... Next or a Do... Loop) are only accessible inside that block of code.

a) \( V \) is a variable defined in Block B
b) \( V \) is a returned value to a method in Block B

vii. Class to Interface

To recall; Interfaces are used when you have two or more classes that represent the same public members, but carry out different purposes.

a) Class C1 implements interface

viii. Method to Interface

a) Method M1 implements interface

3.2.2 Change Impact Analysis in VB.Net

Every programming language has its uniqueness, in normal structured programming language one thinks in terms of inputs, functions and outputs. This differs in OOP
language especially in VB.Net. A message will be passed to an object to request an operation on the object. Objects consist mainly of two parts: methods and data members [Li, L. 1998]; as known, the methods specify the operations allowed on the object's private data, while the data members specify the state information for the object. Moreover, the method and the data member are considered to be a class member. So, referring to [Li, L. 1998] when a class member changes it might affect (impact) other classes through a message passing, inheritance, and different ways.

Mainly we will look in our analysis at the Class which is composed of member functions and variables.

Looking at the way classes interact with the methods and data members we can conclude a direct relationship as follows:

There is a direct relationship R between class A and B (ARB) if A and B have one of the three kinds of relationships: Containment, Use, or Inheritance. [Li, L. 1998]

According to [Li, L. 1998] in the Direct relationship were defined as follows:

- Containment: Class A contains class B if B declared as a class member of A.

- Use/Reference: there are different ways that the Use/reference can be formed:
  
  i. Containment: If class A contains class B, then class A uses Class B. If A contains B by reference that means that A contains a reference to B. B's life span can be longer than A's.

  ii. Classes passed in as method parameter: If a method m of class A takes parameters P1...Pn, we say class A uses each pi, i=1...n, and m is in the reference sets of each of Pi. Pi can be any class and type.

  iii. Classes referenced in the left hand side of assignment: if class A or one of its members is specified in the left hand side of the assignment statement, A or its member is defined by all the variables on the right and side. Thus, class A (or its member)
belongs to the reference set of all those variables on the right hand side of the equation.

iv. **Return type of method:** the return type of a method \( m \) is defined by \( m \). \( m \) belongs to the reference set of this return type. Since the parameters may not be used in the body, and their effect may not direct impact the return type, we will not be considering the return type to be defined by these parameter types. If the return type is defined by a parameter, it will show up in the analysis of this method body.

v. **Variable declared in a method:** any variable that is referenced in the method \( m \) can be considered to be used by \( m \) and can be put into the reference set of \( m \).

- Inheritance relationship: class \( A \) inherits from Class \( B \) if \( B \) is declared as a super class of \( A \).

### 3.2.3 Types of Changes and their relationship

In this section we will list from the syntactic point of view, the typical types of changes that may be made by VB.Net object oriented programs, [Salem, H. 2004] listed the changes in typical Object Oriented Programs, although VB.Net is pure Object Oriented language it differ a little from typical OOP. The list below will include the common changes and their relationship:

i) System level change:
   a. Add super class
   b. Delete super class
   c. Add sub class
   d. Delete sub class
   e. Add aggregated class

42
f. Delete aggregated class

h. Add interface

g. Change inheritance type
   1. Change from public inheritance to private inheritance
   2. Change from private inheritance to public inheritance

i. Delete interface

j. Add methods’ prototype to interface

k. Delete methods’ prototype from interface

l. Add/edit a database connection

m. Delete a database connection

n. Add/change a shared variable

o. Delete a shared variable

ii) Class level change

   a. Add member

   b. Delete member

   c. Define/Redeﬁne member

   d. Change member/method

      1. Change member scope

         a. Change from public to private

         b. Change from public to protected

         c. Change from protected to public

         d. Change from protected to private

         e. Change from private to public

         f. Change from private to protected

         g. Change from private to friend

         h. Change from public to friend

         i. Change from friend to public

         j. Change from friend to private

         k. Add/delete a shared deﬁnition

      2. Change method
a. Protocol change
   i. Name change
   ii. Parameter change
   iii. Return type change
b. Implementation change
   iv. Interface change

3. Change Data member
   a. Add data declarations
      v. Delete data declaration
      vi. Add data definition
      vii. Delete data definition
      viii. Change data declaration
         1. Change data type
         2. Change data name
      ix. Change data definition

4. Function implementation change
   e. Add/delete an external data use
   f. Add/delete an external data update
   g. Add/delete/change a method call
   h. Add/delete a sequential segment
   i. Add/delete/change a branch/loop
   j. Change a control sequence
   k. Add/delete/change local data
   l. Change sequence segment

According to [Salem, H. 2004] *inheritance* will be assigned the greatest impact power, with the *containment* relationship as medium and *use* relationship the least. That is right because the super classes defines the sub classes behavior. Where any change in the public and protected levels of the super class should impact the sub classes that are in its scope. The containment relationship is considered to have a higher impact than the use relationship because the contained class constructors and destructors are always called by
the container class. Since the coupling between inheritance is much higher than the impact power will have a higher value than the containment and the use ones.

3.3 Change Propagation

Discussing the change propagation in ASP.Net websites means discussing two sets of change propagations, one related directly to the .aspx presentation file including the modules and classes that are defined in the script written Object Oriented Programming Language (VB Scripts) and one related directly to the code behind written in Object Oriented Programming Language (Focusing on VB.Net).
Change propagation in the .aspx file can be classified in to Form-Scope Propagation and Page-Scope Propagation.

- **Form-Scope Change Propagation:**
  Is the propagation inside the .aspx file itself (working only with ASP/HTML tags), inside a Form and Control tags’ scope

  1. Controls
     a. A control is defined in the form scope
     b. An item is defined in a control scope
     c. A control is set to validate another control
     d. A control is set to be used by/ or use another control
     e. A control value (ex: text box value) is used as an input value to a module in the script inside the .aspx file

  2. Global or inherited methods
  3. Defined User Control
  4. Inherited methods (may be included by inherited classes)

- **Page-Scope Change Propagation**
Control dependences, Data dependence, Call dependence and event-based dependence are what we focused on in this section.

1. Page Scope Methods (including shared, public...etc)
2. Page Scope variables (including shared, public...etc)
3. Inherited methods in case of code behind
4. Control value is an input parameter to an inherited method
5. Control Value is an output parameter from a method in code behind
6. Hyperlink to another page/window (pop up windows)

Change propagation in the .aspx.vb file or in a script inside .aspx file can be classified into Inter-class propagation and Intra-class propagation.

- **Inter-class change propagation:**
  As [Salem, H. 2004] stated, it is the propagation inside the method’s class. It can either be in a script inside the .aspx file or in a class inside the .aspx.vb code behind.

1. Method Scope Variables (inside the module in a script or codebehind)
   a. The variable is defined in an assignment statement
   b. The variable is assigned a value read from an input
   c. The variable is an input parameter to a module
   d. The variable is an output parameter from a called module

2. Class Scope or inherited variables
   a. A variable with a modifier “private” is global to all the methods in the class
   b. Inherited variable

3. Class Scope or inherited methods
4. Inherited methods/variables

- **Intra-class change propagation:**
  Data dependency/relationship among different methods and functions inside and outside class. [Salem, H. 2004]

1. Global variables (including shared, public...etc)
2. Global methods
3. Public/protected members of a super class in case of inheritance
4. All public members of any class inside the system
5. Variable is an input parameter to a called method/class (via a message)
6. Variable is an output parameter of a method.
Chapter 4

Computing Ripple Effect of a Web Application

The purpose of this chapter is to give a clear idea about the way we can calculate the ripple effect of an ASP.net website. The computation of a .net website is calculated in different ways depending on the structure of the code as discussed earlier in section 2.5. In all ways we have four fundamental ideas in the computation of ripple effect in .net website that is the local and global for the .aspx page and Inter-class and Intra-class for the .aspx.vb code and the vbscript. Previous work was done by (Sue Black, July 2001) and (Mansour and Salem, July 2004) considering the matrix product as a way to calculate the Ripple effect following certain rules. In this chapter we will consider the same way in calculating the ripple effect but based on websites written in .Net environment. Because of the differences that are found between normal programming languages (either Object Based Programming or Object Oriented Programming) and ASP.Net language, different rules will be followed in creating the matrices of an ASP.net website. This chapter gives a detailed description of what they are and the way they are used to calculate the Ripple Effect, in section 2.5 three different ways in creating a .net website was discussed according to that discussion we will focus on different ways in calculating their ripple effect. Then the description is followed by an example to clarify the use of the ripple effect in practice.

4.1 Web Code Complexity Metric

A complexity measure should be taken into account in any .aspx file. A matrix $C$ can be presented, representing a complexity measurement similar to McCabe's cyclomatic complexity for the controls in our code indexpage.aspx. Cyclomatic complexity was introduced by Thomas McCabe in 1976 where he published a paper
stating that the code complexity should be defined by its control flow. Since that time
other researchers identified different ways of measuring complexity. An automated
program called “VB LAW” [Progress Technologies Limited, 2006] computes the
cyclomatic complexity of a Visual Basic code. Based on that we propose in this section a
way to calculate the complexity of a .aspx page that contains no scripts also applicable
for .aspx.vb files called “BABA’s Complexity”.

According to the “indexpage.aspx” file we can find no decision point written in the code,
that’s why to calculate the complexity of a certain .aspx file that contains web controls
we have to be knowledgeable about how the whole form is working. To calculate the
complexity of our code ($\delta$) we have to analyze the code as follows and use BABA’s
Complexity Measurement that works as follows:

$$\delta = \sum \alpha + 1$$

Where $\alpha$ refers to an occurrence of a control that makes/triggers an event (Hyperlink,
Submit Button, OnClick Buttons...etc). For VB.Net/VBscript code $\alpha$ represents each
type of a control flow or decision point (If ... then... else; while ... end while... etc)
applying it to our code in Fig 4.2:

1. The txt_pass_validator validates the presence of a text
   value in the login text box, it can be written as :

   \[
   \text{if (text_password.text = "\"0\") then}
   \]
   \[
   \quad \text{message (\"})
   \]

   Of course this code is a pseudo code. This validator is
   acting as a decision point thus incrementing the value of
   $\alpha$ by one so $\alpha = 1$

2. The same goes for txt_login_validator so $\alpha = 2$

3. The submit button is always considered as a decision
   point or control flow so add $\alpha = 3$

4. The validation summary can not be considered because
   its value is depending on the value of the validators so it
   can be embedded under the same decision point:

   \[
   \text{if (text_password.text = "\"NULL\") then}
   \]
So the Complexity of Form1 in Fig 4.2 is \( \delta = 3+1= 4 \). The running time of computing Baba’s Complexity metric is \( O(L_i) \), where \( L \) is the number of lines of code in Form or method \( i \).

4.2 Form-Scope Change Propagation

The computation of ripple effect in .aspx file is based on the effect that a change in a control might affect the functionality of another control thus affecting the whole .aspx page. In the following figure we are showing a page called “indexpage.aspx” where a student is asked to enter his Login and Password, if he forgot to write one of them a signal “*” will be triggered to remind him to write in the field and not leaving it blank, a validator summary is responsible to explain the type of error the user made found at the bottom left part of the figure. A script is added to the page for computation purposes.

![Lebanese American University](image)

Fig 4.1 : Login page created in ASP.Net made by .aspx and code behind

The code in Fig 4.2 is the .aspx file called “indexpage.aspx” it shows the exact code that is written in ASP/Html only inheriting a code behind file called “indexpage.aspx.vb” without any script written in it.
Fig 4.2: Code written in ASP/HTML for indexpage.aspx presentation file

The code in Fig 4.2 includes some controls highlighted in Grey and others are not; that is because not all controls will be considered to calculate the Ripple effect of this page, some of them will not affect the structure of the website upon modification; labels,
images, lines, font, marquee and a lot or other control. Only those who might affect the
structure of the website (Highlighted in Grey) will be considered.

Form-Scope Propagation due to a change in a control is based on the following five
conditions derived from dependencies analysis described in chapter 3:

1. An item is defined in a control scope, in Fig 4.2 there is no
such control. But an example about is item “Fall2006” is found
in “SEM” dropdownlist scope.

```
<asp:DropDownList id="Sem" style="Z-INDEX: 103; LEFT:
136px; POSITION: absolute; TOP: 224px" runat="server"
Width="168px" Height="40px">
    <asp:ListItem></asp:ListItem>
    <asp:ListItem Value="Fall2006">Fall2006</asp:ListItem>
    <asp:ListItem Value="Spring2006">Spring2006</asp:ListItem>
</asp:DropDownList>
```

2. A control is set to validate another control. For an example
“txt_login_validator” is set to validate the “txt_login” web
control.

```
<asp:requiredfieldvalidator id="txt_login_validator"
style="Z-INDEX: 105; LEFT: 456px; POSITION: absolute;
TOP: 176px" runat="server" ControlToValidate="txt_login"
ErrorMessage="*" />
</asp:requiredfieldvalidator>
```

3. A control is set to be used by/ or use another control. For an
example the “VS” validation summary is set to use all the
validators’ values in the .aspx to write them in its summary

```
<asp:validationsummary id="VS" style="Z-INDEX: 110;
LEFT: 72px; POSITION: absolute; TOP: 416px"
runat="server" Width="184px" Height="40px">
</asp:validationsummary>
```

4. A control value (ex: text box value) is used as an input value to
a module in the script inside the .aspx file. The following text
box is an example where the value of “txt_pass” is sent to a
method in code behind to check whether the password is correct.

```html
<asp:textbox id="txt_pass" style="Z-INDEX: 102; LEFT: 312px; POSITION: absolute; TOP: 248px" runat="server" Width="137px" Height="34px" TextMode="Password">
</asp:textbox>
```

5. Page-Scope or inherited methods called by control events. The “submit” button web control is calling a method called “button_click” inherited from code behind file sending it the values of the txt_login and txt_pass to check if they match a student’s login and password or not.

```html
<asp:button id="submit" style="Z-INDEX: 104; LEFT: 320px; POSITION: absolute; TOP: 304px" runat="server" Width="136px" Height="32px" ForeColor="#004000" Text="Submit">
</asp:button>
```

As written earlier only the controls that follow the above rules will be considered. Using these rules we can develop a vector $V_F$ representing a 0-1 vector that represents the control definition in Form F. Controls that satisfy any of the above conditions will be denoted by “1” and those which do not by “0”. We shall use a short name for each control found in the Form (for example: txt_login will be written as “tlg”), note that labels and lines that are not interacting with the controls or has nothing to do with the functionality of the website will not be taken into consideration. So the dimension of the vector $V_F$ will be equal to the number of variables/controls found in the .net page that has direct relationship with the page’s functionality. Based on this, the vector $V_{FJ}$ will be represented as follows:

$$
V_{FJ} = \begin{bmatrix}
    tlg & tpv & tp & sub & tlv & vs & wl \\
    1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

tlg represents txt_login

tpv represents txt_pass_validator
tp represents txt_pass
sub represents submit
tlv represents txt_login_validator
vs represents validationsummary "VS"
wl represents Welcome label

The worst case running time for determining \( V_{F1} = O(\beta_1 L_F) \) where \( L_F \) is the number of lines found in Form i. And \( \beta_1 \) is the time required to determine whether an element belongs to the controls defined in the five conditions discussed before.

A 0-1 direct impact matrix \( S_F \) can be produced to show which control value may propagate to other controls within the form F. The rows and columns \((r, c)\) of \( S_F \) represent each individual occurrence of a control. \((r, c)=1\) indicates a propagation from row \( r \) to column \( c \), a change in \( r \) will directly affect \( c \) according to one or more of the above five mentioned conditions. Note that \( S_F \) is not bidirectional (symmetric) matrix. A brief description of this propagation will be shown after representing the matrix \( S_F \) below:

\[
S_{F1} = \begin{pmatrix}
tlg & tpv & tp & sub & tlv & vs & wl \\
1  & 0  & 0  & 1  & 1  & 0  & 0  \\
0  & 1  & 0  & 1  & 0  & 1  & 0  \\
0  & 1  & 1  & 1  & 0  & 0  & 0  \\
0  & 1  & 0  & 1  & 1  & 0  & 0  \\
0  & 0  & 0  & 1  & 1  & 1  & 0  \\
0  & 0  & 0  & 0  & 0  & 1  & 0  \\
0  & 0  & 0  & 0  & 0  & 0  & 1 \\
\end{pmatrix}
\]

We observe in the above matrix that the value of \( tlg \) will propagate to itself of course and to \( tlv \) because \( tlv \) is a validator web control that checks if there is a value in \( tlg \) or not so it depends on its value to function. While the value of \( tlg \) will also propagate to the button \( sub \) because \( sub \) is sending the value in \( tlg \) to a method in code behind file to check whether that login value is available in the database or not. The running time for
determining $S_F$ is $O(\gamma L_{F_i})$ where $L_i$ is the number of lines in Form $F_i$ and $\gamma$ is the time to determine the propagation between controls in Form $F_i$.

4.3 Page-Scope Change Propagation

Propagation across forms and scripts in the .aspx presentation file is called Page-Scope Change Propagation. A change to a control can propagate to other method in a script or to another control in another form if it falls under those three conditions (derived from dependencies analysis described in chapter 3):

1- The control value works as an input parameter to a scripted method in the .aspx file

2- The control’s value is a returned value from a method in a script under a certain event.

3- A control is setting a value or working with another control in another form.

The matrix $X_{F_i}$ will be made from number of columns equal to the number of forms and methods within the page and the number of rows is equal to the number of controls used. Thus, $X_F$ for our example is given by:
\[ X_{F1} = \begin{pmatrix} \text{tlg} & 0 & 0 \\ \text{tpv} & 0 & 0 \\ \text{tp} & 0 & 0 \\ \text{sub} & 0 & 0 \\ \text{tlv} & 0 & 0 \\ \text{vs} & 0 & 0 \\ \text{wl} & 0 & 1 \end{pmatrix} \]

The running time for determining \( X_{F1} = O(\beta_2 L_p) \) where \( L_p \) is the number of lines found in a webpage. And \( \beta_2 \) is the time required to determine whether an element belongs to the controls defined in the three conditions discussed before.

As we notice there is no propagation from any control in \( F1 \), because the Page-Scope Change Propagation involves flow of program change across forms and scripts. Whereas the label “Welcome” takes its value from the page_load (PL) script. The Page-Scope Change Propagation of all controls in \( F1 \) can be found by finding the Boolean product of \( S_{F1} \) and \( X_{F1} \). In our case:

\[
S_{F1}X_{F1} = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1 \\
0 & 1
\end{pmatrix} = \begin{pmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix}
\]

We used the Boolean matrix product of \( S_{F1} \) and \( X_{F1} \) to maintain consistency with Form-Scope Change Propagation computation which is also Boolean. The running time for determining the matrix product \( S_{F1} X_{F1} \) is \( O(\theta_{\text{max}} \times \Omega) \) where \( \theta_{\text{max}} = \max\{\theta_i\} \) for \( i=1 \) to \( \Omega \), where \( \theta \) is equal to the number of variables/controls found in the .net page that has direct relationship with the page’s functionality and \( \Omega \) is the number of forms/methods per page. In order to indicate the amount of propagation from web controls in a Form,
assume Form1 to another Form or method in a script we should find the standard matrix product of \( V_{F1} \) and \( S_{F1}X_{F1} \) as follows:

\[
V_{F1}S_{F1}X_{F1} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 0
\end{pmatrix} = \begin{pmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix}^{F1 \text{ PL}}
\]

This result states that there are zero propagation from Form1 to Form1 and one propagation from Form1 to Page_Load (PL) script. Note that the only label “Welcome” is propagating to Page_Load. Of course if the form had more forms or scripts then the result will be different. The running time for determining \( V_{F1}S_{F1}X_{F1} = O(\theta_{F1} \times \Omega) \).

The complexity measurement for the indexpage.aspx (Form Scope) is \( \alpha_{F1} = 4 \) and the complexity measurement of the script page_load \( \alpha_{PL} = 1 \). Finding the product of \( V_{F1}S_{F1}X_{F1} \) and Complexity matrix \( C \) will represent the complexity-weighted total control-change propagation for Form1:

\[
V_{F1}S_{F1}X_{F1}C = \begin{pmatrix}
0 & 1
\end{pmatrix} = \begin{pmatrix}
4 \\
1
\end{pmatrix}
\]

The running time for determining \( V_{F1}S_{F1}X_{F1}C = O(\Omega) \). The maximum value that \( V_{F1}S_{F1}X_{F1}C \) can have is \( \sum_{i=1}^{\Omega} \theta_{i} \delta_{i} \). We normalize the number obtained by the product \( V_{F1}S_{F1}X_{F1}C \) by dividing it with the maximum value that can be obtained which is \( \sum_{i=1}^{\Omega} \theta_{i} \delta_{i} \) to give the mean complexity-weighted control-change propagation per
control in Form1. In our code the \( \sum_{i=1}^{n} \theta_i \delta_i = 7 \times 4 + 1 \times 1 = 29 \). Therefore, ripple effect of Form1 is defined as:

\[
RE_{FL} = ( V_{FL} \cdot S_{FL} \cdot X_{FL} \cdot C ) / \sum_{i=1}^{n} \theta_i \delta_i = 1/29 = 0.034
\]

The logical stability measure for Form1 is defined to be the reciprocal of the ripple effect value plus one. The \( RE_{FL} \) is 0.034, so the Logical stability is equal to the following

\[
LS = 1 / (RE+1)
\]

So in our code the \( LS = 1 / (0.034 + 1) = 0.96 \).

The \( RE \) value in any page is \( 0 \leq RE \leq 1 \) and Logical stability \( 0.5 \leq LS \leq 1 \). These numbers of LS and RE should be read as relative and not absolute, so that to compare the RE and LS of different forms in different pages or websites.

From the above result we can conclude that the ripple effect of any .aspx file that contains one form and no scripts (i.e. following the first way in creating .aspx file) is always zero and its logical stability is always 1.

### 4.4 Inter-class Change Propagation

Calculating the Inter-class change propagation means that the webpage contains a code behind file or a script written in visual basic (in the .net environment). That is the developer used either the second way of creating web pages or the third one (discussed in section 2.5). The computation of ripple effect is based on the effect where a change to a variable will affect the rest of the webpage.

```
1 Imports System.Data.OleDb
2 Public Class indexpage
3 Inherits System.Web.UI.Page
4 Public Shared user_ID As Long
5 Dim cn As OleDbConnection
```
Fig 4.3: Code written in VB.Net for code behind indexpage.aspx.vb

In above example we can read the codebehind file of indexpage.aspx presentation file, mainly it is made from one class called “indexpage” and two methods one called “page_load” that is called once the page is loaded, and another called “submit_click” that is called once the submit button in index.aspx is clicked. Simply, this code creates a database connection and opens it to be accessed through a query called “cmd” in line 11 in page_load method. Then when the user enters his login and password and clicks on the submit button the submit_click method will be called to execute the query in line 14 and checks whether a user is found (in line 16) then the page will be redirected to another one called personal.aspx or not in line 19 so the user will have a note saying that the user_login and password are incorrect and he has to check them and write them again. Clearly in the page_load method in line 9 we are defining the connection and we are using it in line 10 where we are opening the connection, it is also used in line 11 to create
a query called “cmd” so any change that might happen in line 9 will affect line 10 thus affecting the query “cmd” in line 11. Clearly propagation takes place from definitions to uses of variables and via assignments.

Based on the work [Salem, 2004], Inter-class ripple computation due to a change in a variable will mostly be based on the following six conditions derived from dependencies analysis described in chapter 3:

1. The variable is defined in an assignment statement. For example ‘cmd’ in :

   ```
   11 cmd = New OleDbCommand("select * from Login where UserLogin = "+ txt_login.Text.Trim + " and UserPassword = "+
   & txt_pass.Text.Trim + ", cn")
   ```

2. The variable is assigned a value which is read as an input. For example ‘dr’ in line 15:

   ```
   15 dr.Read()
   ```

3. The variable is calling a function or a method that is essential to the rest of the program to work. Example ‘cn’ in

   ```
   10 cn.Open()
   ```

4. The variable is an input parameter to a method. An example not found in the above figure can be ‘ccounter’ in:

   ```
   Public double calculate ( double ccounter )
   ```

5. The variable takes a returned value from a called method. An example can be ‘dr’ in :

   ```
   12 dr = cmd.ExecuteReader
   ```

6. The variable is in Class-Scope, shared or inherited variable. The same example can be taken from condition 6 because cmd is a global variable defined globally for all the methods in class indexpage.

   ```
   2 Public Class indexpage
   3 Inherits System.Web.UI.Page
   4 Public Shared user_ID As Long
   5 Dim cn As OleDbConnection
   6 Dim cmd As OleDbCommand
   ```
Of course each programming language has its own visibility rule for its variables, in VB.Net if a variable is defined in a block scope then it is not visible outside that scope. When creating the matrix \( V_m \) we will be working under the VB.Net visibility rules. Previous work was done covering another object oriented language “Java”, it is similar to VB.net but each has its uniqueness. Intuitively any global or shared values on the right hand side of assignments should count. Based on the conditions above we will develop a matrix \( V_{load} \) and move on in calculating the page_load ripple effect. Therefore the matrix \( V_{load} \) will be:

\[
V_{load} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

\( cn_{98} \) represents \( cn = New \ OleDbConnection(\text{"Provider...etc defined in line 9} \)
\( cn.o \) represents \( cn.Open() \) in line 10
\( cmd \) represents \( cmd = New \ OleDbCommand(\text{"select * from...etc in line 11} \)
\( txt_i \) represents the use of \( \text{txt_login in line 11} \)
\( txt_p \) represents the use of \( \text{txt_pass in line 11} \)
\( cn_{11i} \) represents the use of \( cn \) in line 11 as a parameter
\( dr \) represents \( dr \) executing \( \text{cmd query "dr = cmd.ExecuteReader" in line 12} \).

The variables are taken in the order of appearance in the page_load method, if two variables appeared in the matrix having the same name then they will be differentiated by a “n” or a “d” meaning either used or defined followed by the line number. A 0-1 direct impact matrix \( S_{load} \) can be produced to show which variable may propagate to other variables within the method page_load. The rows and columns of \( S_{load} \) will represent each individual occurrence of a variable. Propagation will be shown from row \( i \) to column \( j \). A brief description of this propagation will be shown after representing the matrix \( S_{load} \):
\[
S_{load} = 
\begin{pmatrix}
\text{cn}_9 & \text{cn}_o & \text{cmd} & \text{txt}_l & \text{txt}_p & \text{cn}_n & \text{dr} \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

We observe in the above matrix that the value of \text{txt}_l will propagate to itself of course and to \text{cmd}. While the value of \text{cmd} will propagate to \text{dr} where \text{dr} will be used to execute and read the value returned from the query. So the \text{txt}_l will also propagate to \text{dr}. We can notice that \(S_F\) is both reflexive and transitive; that is every variable occurrence is assumed to propagate to itself and if a variable \(v\) propagates to variable \(s\) and variable \(s\) propagates to variable \(n\) then variable \(v\) will also propagate to variable \(n\). as mentioned earlier, \(S_F\) it shows how a variable's value can reach or affect another variable in the code.

### 4.5 Intra-class Change Propagation

Propagation across classes and methods in a script (in .aspx) or code behind file .aspx.vb is called \textit{Intra-class change propagation}. Based on the work [Salem, 2004] a change to a variable can propagate to other method in a script or to another other method in a class found in code behind if it falls under one of the following conditions:

i. The variable is an inherited, shared or a global variable. For example

```vbnet
Public Class indexpage
    Inherits System.Web.UI.Page
    Public Shared user_ID As Long
    Dim cn As OleDbConnection
    Dim cmd As OleDbCommand

    . .
    . .
    . .
    dr = cmd.ExecuteReader
```
The *cmd* query command is global to all the methods in class indexpage, a change in *cmd* in any method might affect *dr*.

ii. The variable is an input parameter in an Intra-class message. An example can be counter1 in:

\[
\text{Average} = \text{Grades} . \text{calculate (counter1)}
\]

iii. The variable or object is returned by a method to another method in a different class. An example can be *age* in:

\[
\text{Return (age)}
\]

In the example indexpage.aspx.vb we can analyze the method *page_load* according to the above conditions. For example, the definition of a new connection to a database *cn* is a global variable, so any change in that definition will affect all the methods in the class. The same thing goes for the query command *cmd*. While *dr* is defined globally then its execution in *page_load* will affect the values read in *submit_click* method. Because the values returned in *dr.Read()* in line 16 depends directly on the execution of *cmd* in line 12 "*dr = cmd.ExecuteReader*". We can represent the propagation of these variable through the class using a 0-1 matrix *X_load* :

\[
X_{load} = \begin{bmatrix}
\text{cn49} & 0 & 1 \\
\text{cn0} & 0 & 1 \\
\text{cmd} & 0 & 1 \\
\text{txt_1} & 0 & 0 \\
\text{txt_p} & 0 & 0 \\
\text{cnall} & 0 & 0 \\
\text{dr} & 0 & 1
\end{bmatrix}
\]

Intra-class change propagation involves the flow of program/variable change across classes, that is why the column *P_load* is all zeros. By finding the Boolean product of *S_load* and *X_load* we can find the Intra-class change propagation of all variables in *page_load*. Therefore the *S_load X_load* will be:

63
\[ S_{load} \times X_{load} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \begin{pmatrix}
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
\end{pmatrix} = \begin{pmatrix}
0 & 3 \\
0 & 2 \\
0 & 2 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
\end{pmatrix}\]

Now in order to indicate the amount of propagation from variables in the load method, we should find the standard matrix product of \( V_{load} \) and \( S_{load} \times X_{load} \) as follows:

\[ V_{load} \times S_{load} \times X_{load} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
\end{pmatrix} \begin{pmatrix}
0 & 3 \\
0 & 2 \\
0 & 2 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
\end{pmatrix} = \begin{pmatrix}
0 & 11 \\
\end{pmatrix}\]

A complexity measure should also be taken into account in any .aspx.vb. A matrix C can be presented, representing a complexity measurement. Complexity measurement can be presented in different ways, one of which is the McCabe cyclomatic complexity presented by Thomas McCabe, the cyclomatic complexity represents the static complexity of the program. [Salem, H. 2004] represented another kind of complexity measurement based on using weighted sum of metrics for each class/method.

In our example we will use our method method “BABA’s Complexity Measurement”. Because we used the BABA’s Complexity Measurement in .aspx presentation file, we
will keep this consistency and continue using BABA’s complexity measurement to represent a matrix C as follows:

$$C = \begin{pmatrix}
P_{load} \\
Sub\_click
\end{pmatrix}
\begin{pmatrix}
1 \\
3
\end{pmatrix}
$$

The product of $V_{load}S_{load}X_{load}$ and C is:

$$V_{load} \cdot S_{load} \cdot X_{load} \cdot C = \begin{pmatrix}
0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
1 \\
3
\end{pmatrix} = 33$$

We normalize this number by dividing it by $\sum_{i=1}^{n} \theta_i \delta_i$ to give the mean complexity-weighted control-change propagation per variable in $Page\_load$. In our code, $\sum_{i=1}^{n} \theta_i \delta_i$ is equal to 37. Therefore, ripple effect of page_load is defined as:

$$RE_{load} = \frac{(V_{load} \cdot S_{load} \cdot X_{load} \cdot C)}{\sum_{i=1}^{n} \theta_i \delta_i} = \frac{33}{37} = 0.89$$

The logical stability measure for $Page\_Load$ is defined to be the reciprocal of the ripple effect value plus one. In our code the RE is 0.89 so the logical stability will be:

$$LS = \frac{1}{RE+1} = 1/1.89 = 0.52$$

These numbers of LS and RE should be read as relative and not absolute, so that to compare the RE and LS of different forms in different pages or website.

### 4.6 Finalizing the Ripple effect calculation

The calculation of the Ripple effect of a website does not stop when calculating the intra/global and intra/Intra-class ripple effects. It is expanded more because a website is not made from a single page it can be created by multiple pages each can follow a different way in development, the figure below will illustrate a website made from three
pages each following a different way in calculation and next based on this illustration we will show what should be computed more to give the total RE and Logical Stability of a website. The illustration is the following:

![Diagram of ASP.Net pages with file types and arrows]

Fig 4.5: A three paged website each developed in a different way

We can notice that page 1 is made from two files one is the .aspx file and the second is the code behind file .aspx.vb in order to calculate the ripple effect of page one we have to calculate the RE of the .aspx file as shown in 4.2 and 4.3 then calculating the RE as shown in 4.4 and 4.5 but following the arrow number 2 in the fig 4.5 we will find that it is aiming on the code behind that might contain more than one method so the RE of the whole .aspx.vb file will be:

$$\frac{1}{n} \sum_{i=1}^{n} \left( \frac{Vmi \cdot Smi \cdot Xmi \cdot C}{\sum_{i=1}^{n} \theta_i \delta_i} \right)$$

Where m= method and n=number of methods in the code behind.

The same goes for arrows number 3, 5 and 6. The total ripple effect of the whole .aspx page which is

66
\[
\frac{1}{l} \sum_{t=1}^{l} \left( V_{ki} \cdot S_{ki} \cdot X_{ki} \cdot C \right) \sum_{\alpha=1}^{n} \theta_{\alpha} \delta_{t}
\]

Where "\(k\) = a method or a form" and "\(l\) = number of total methods and forms" in the .aspx file. The total ripple effect of the whole page (Page1 or Page2 or Page3). In case of code behind we can not rely on creating a 0-1 direct impact matrix \(Se\) that can be produced to show the propagation between the .aspx and .aspx.vb because it will always be full of ones. The .aspx and .aspx.vb are strongly connected one cannot work without the other, that's why we will follow the second way where the total Ripple effect of the .net page as a whole will be the sum of the ripple effect of .aspx and .aspx.vb divided by two and its reciprocal will be the logical stability.

After finding the ripple effect of each page, the ripple effect and logical stability of the whole website is calculated:

\[
RE_{website} = \frac{1}{w} \sum_{i=1}^{w} RE_{pi}
\]

Where \(w\) is the number of pages in a website and \(P\) is a single web page.

We note that this expression is an average weighted sum of REs, where the weight is the complexity of each page.

According to the above rule, assuming the web pages’ ripple effect as:

\[
RE_{p1} = 4 \\
RE_{p2} = 6 \\
RE_{p3} = 5
\]

Then the Ripple effect of the whole website will be:

\[
(4+6+5)/3 = 5
\]

The logical stability will therefore be: \(1/(5+1) = 0.16\). These numbers of LS and RE should be read as relative and not absolute, so that to compare the RE and LS of different websites.
Summary of all complexities in computing ripple effect of a webpage is illustrated in Table 4.1

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determining Vector $V$</td>
<td>$O(\beta_1 L_{F_i})$</td>
<td>$L_{F_i} =$ number of lines found in Form $i$. $\beta_1 =$ time required to determine whether an element belongs to the controls defined in the five conditions discussed in 4.2.</td>
</tr>
<tr>
<td>Constructing matrix $S$</td>
<td>$O(\gamma L_{F_i})$</td>
<td>$\gamma =$ time to determine the propagation between controls in Form $F_i$.</td>
</tr>
<tr>
<td>Constructing matrix $X$</td>
<td>$O(\beta_2 L_p)$</td>
<td>$L_p =$ number of lines found in a webpage. $\beta_2 =$ time required to determine whether an element belongs to the controls defined in the three conditions discussed in 4.3.</td>
</tr>
<tr>
<td>Product matrix $S.X$</td>
<td>$O(\theta_{max} \times \Omega)$</td>
<td>$\theta_{max} = \max{ \theta_i }$ for $i=1$ to $\Omega$, where $\theta$ is equal to the number of variables/controls found in the .net page that has direct relationship with the page’s functionality. $\Omega =$ number of forms/methods per page.</td>
</tr>
<tr>
<td>Product matrix $V.S.X$</td>
<td>$O(\theta_{F_i} \times \Omega)$</td>
<td>$\theta$ is equal to the number of variables/controls found in Form/method.</td>
</tr>
<tr>
<td>Complexity Matrix $C$</td>
<td>$O(L_i)$</td>
<td>$L_i =$ the number of lines of code in Form or method $i$.</td>
</tr>
<tr>
<td>Product Matrix $V.S.X.C$</td>
<td>$O(\Omega)$</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

A website example

Calculating the ripple effect of a website in a complex process and tedious, automating the computation is much more desirable. But a problem might be faced in the automation, it is time consuming. Yau and Chang gave an example of a two thousand line program’s stability measure taking thirteen hour of CPU time to be computed. A lot of problems where encountered during the automation of Inter-class change propagation stage, where program slicing was used. In this chapter, I developed a three pages website called TestLogin where it works as follows (Fig 5):

Fig 5: Three pages that forms a website called Testlogin
In (Fig 5) we can find an illustration of a website that is called “Testlogin” it is made from three pages:

- Page1: “indexpage” page where a student enters his login and password
- Page2: “personal” page where a student can check his course schedule in a selected semester
- Page3: “grades” page where a student checks his grades in a selected semester.

The ripple effect of each page will be calculated alone, some of which is developed using only .aspx file (that means only ASP/HTML and scripts without codebehind) and others using code behind files with .aspx file of course. After calculating the ripple effect of each page we will conclude by calculating the total ripple effect of the whole website.

5.1 Computing Ripple Effect for the “Indexpage”

Before we jump in to calculating the Ripple Effect of the “Indexpage”, we have to study the structure of this page first, so that to know how are we going to move on in the process as discussed in chapter 4. Looking at the code of the Indexpage we can notice that there is two main files “indexpage.aspx” and “indexpage.aspx.vb” so this page uses a code behind file to work. In the next section we will calculate the ripple effect of each page alone then calculating the total Ripple Effect of the indexpage at the end of section 5.1.2.
Fig 5.1 (b): Code written in ASP/HTML for indexpage.aspx presentation file
5.1.1 Computing Ripple Effect for the Indexpage.aspx file

We can find that the indexpage.aspx (Fig 5.1 (b ) presentation file contains no scripts and only one form. So the computation should proceed as follows:

In chapter3 we concluded a rule that is: the ripple effect of any .aspx file that contains one form and no scripts (i.e: following the first way in creating .aspx file) is always zero and its logical stability is always 1.

The Ripple Effect is,

\[ \text{RE}_{\text{index.aspx}} = 0 \]

The logical stability is,

\[ \text{LS}_{\text{index.aspx}} = \frac{1}{1 + \text{RE}_{\text{index.aspx}}}= 1 \]

5.1.2 Computing Ripple Effect for the Indexpage.aspx.vb file

We can find that the Login.aspx.vb is made from one class called indexpage and this class contains two methods, the first one is loaded automatically when the webpage is load called “page_load” and the second one is called when the user clicks on the “submit” button in the indexpage. Fig 5.1.2 represents the code behind file called “indexpage.aspx.vb”:

```vbnet
Imports System.Data.OleDb

Public Class indexpage
    Inherits System.Web.UI.Page
    Public Shared user_ID As Long
    Dim cn As OleDbConnection
    Dim cmd As OleDbCommand
    Dim dr As OleDbDataReader
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\inetpub\wwwroot\TestLogin\test.mdb;")
        cn.Open()
        cmd = New OleDbCommand("select * from Login where UserLogin = '" & txt_login.Text.Trim & "' and UserPassword ='"
```

72
Fig 5.1.2: Code written in VB.Net for code behind indexpage.aspx.vb

To start calculating the Ripple effect of this file we have to calculate the Ripple effect of the two methods individually.

5.1.2.1 Computing Ripple Effect for the “page_load” method

Computing the ripple effect of this method is made in several steps, they are as follows:

Step1: make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance

cn represents cn = New OleDbConnection("Provider...etc defined in line 9

cn.0 represents cn.Open() in line 10

cmd represents cmd = New OleDbCommand("select * from...etc in line 11

txt_l represents the use of txt_login in line 11

txt_p represents the use of txt_pass in line 11

cn all represents the use of cn in line 11 as a parameter

dr represents dr executing the cmd query “dr = cmd.ExecuteReader” in line 12.
**Step 2:** Form a Boolean matrix for each method.

\[
V_{load} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

**Step 3:** Construct a matrix \( S \) for page\_load method

\[
S_{load} = 
\begin{pmatrix}
\text{cn}_{d9} & \text{cn}_o & \text{cmd} & \text{txt}_l & \text{txt}_p & \text{cn}_{u11} & \text{dr} \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 
\end{pmatrix}
\]

**Step 4:** Construct a matrix \( X \) for page\_load method

\[
X_{load} = 
\begin{pmatrix}
\text{P. load} & \text{sub-click} \\
\text{cn}_{d9} & 0 & 1 \\
\text{cn}_o & 0 & 0 \\
\text{cmd} & 0 & 1 \\
\text{txt}_l & 0 & 0 \\
\text{txt}_p & 0 & 0 \\
\text{cn}_{u11} & 0 & 0 \\
\text{dr} & 0 & 1 
\end{pmatrix}
\]
Step 5: Find the Boolean product of $S$ and $X$

\[
S_{load} \cdot X_{load} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
0 & 0 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix} =
\begin{pmatrix}
0 & 2 \\
0 & 1 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix}
\]

Step 6: Find the Boolean product of $V$ with $S$ and $X$

\[
V_{load} \cdot S_{load} \cdot X_{load} = (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1) \begin{pmatrix}
0 & 2 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1
\end{pmatrix} = (0 \ 8)
\]

Step 7: Construct the complexity matrix $C$

\[
C = \begin{pmatrix}
P_{load} \\
Sub_{click}
\end{pmatrix}
\]

\[
P_{load} = 1 \\
Sub_{click} = 3
\]

Step 8: Find the product matrix of $C$ with $V$, $S$ and $X$

\[
V_{load} \cdot S_{load} \cdot X_{load} \cdot C =
\begin{pmatrix}
0 & 8 \\
3 & 3
\end{pmatrix}
\]

\[
= 24
\]

75
**Step9**: Divide this figure by $\sum_{i=1}^{\alpha} \theta_{ij}$ to get the RE and find the reciprocal of RE to get the logical stability.

$$RE_{load} = (V_{load} \cdot S_{load} \cdot X_{load} \cdot C) / \sum_{i=1}^{\alpha} \theta_{ij} = 24/37 = 0.64$$

The logical stability measure for Page_Load is defined to be the reciprocal of the ripple effect value. In our code the RE was 4.7 so the logical stability will be:

$$LS_{load} = 1 / (1 + RE) = 1/1.64 = 0.61$$

### 5.1.2.2 Computing Ripple Effect for the "submit_Click" method

Computing the ripple effect of this method is made in several steps, they are as follows:

**Step1**: make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance

We have:

- dr, dr.has, user_ID, dr(0), txt_login, txt_pass, txt_label.visible, text_label.text,
- dr.close, cn.close

- dr.has will be presented as dr.h
- txt_login will be presented as tlg
- txt_pass will be presented as tp
- txt_label.visible will be presented as tlv
- text_label.text will be presented as tlt
- dr.close will be presented as dr.c
- cn.close will be presented as cn.c

**Step2**: Form a Boolean matrix for each method.

$$V_{sub} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}$$
**Step 3:** Construct a matrix $S$ for submit_click method

$$S_{sub} = \begin{pmatrix} dr & dr.h & user & dr(0) & tlh & tp & tlv & tlt & dr.c & cn.c \\ dr & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ dr.h & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ user & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ dr(0) & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ tlh & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ tp & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ tlv & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ tlt & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ dr.c & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ cn.c & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**Step 4:** Construct a matrix $X$ for submit_click method

$$X_{sub} = \begin{pmatrix} P \text{load sub-click} \\ dr & 1 & 0 \\ dr.h & 0 & 0 \\ user & 0 & 0 \\ dr(0) & 0 & 0 \\ tlh & 1 & 0 \\ tp & 1 & 0 \\ tlv & 0 & 0 \\ tlt & 0 & 0 \\ dr.c & 1 & 0 \\ cn.c & 1 & 0 \end{pmatrix}$$
Step 5: Find the Boolean product of $S$ and $X$

\[
S_{sub} X_{sub} = \begin{pmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
\end{pmatrix} = \begin{pmatrix}
2 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 0 \\
2 & 0 \\
2 & 0 \\
\end{pmatrix}
\]

Step 6: Find the Boolean product of $V$ with $S$ and $X$

\[
V_{sub} S_{sub} X_{sub} = \begin{pmatrix}
2 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 0 \\
2 & 0 \\
2 & 0 \\
\end{pmatrix} = \begin{pmatrix}
6 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 0 \\
2 & 0 \\
2 & 0 \\
\end{pmatrix}
\]

78
**Step 7:** Construct the complexity matrix \( C \)

\[
C = \begin{pmatrix}
P_{load} & 1 \\
Sub_{click} & 3
\end{pmatrix}
\]

**Step 8:** Find the product matrix of \( C \) with \( V, S \) and \( X \)

\[
V_{sub} \cdot S_{sub} \cdot X_{sub} \cdot C = \begin{pmatrix}
6 & 0 \\
3 & 0
\end{pmatrix}
= 6
\]

**Step 9:** Divide this figure by \( \sum_{i=1}^{\Omega} \theta_i \delta_i \) to get the \( RE \) and find the reciprocal of \( RE \) to get the logical stability.

\[
RE_{sub} = \frac{(V_{sub} \cdot S_{sub} \cdot X_{sub} \cdot C)}{\sum_{i=1}^{\Omega} \theta_i \delta_i} = \frac{6}{37} = 0.16
\]

The logical stability measure for `submit_click` is defined to be the reciprocal of the ripple effect value. In our code the \( RE \) was 1.2 so the logical stability will be:

\[
LS_{sub} = \frac{1}{RE_{sub} + 1} = \frac{1}{1.16} = 0.86
\]

**Step 10:** Calculating the ripple effect of the `indexpage.aspx.vb` file, we have to follow the following rule:

\[
\frac{1}{n} \sum_{i=1}^{n} \frac{(Vmi \cdot Smi \cdot Xmi \cdot C)}{\sum_{i=1}^{\Omega} \theta_i \delta_i}
\]
\[ RE_{\text{indexpage.aspx.vb}} = \frac{0.61 + 0.16}{2} = 0.38 \]
\[ LS_{\text{indexpage.aspx.vb}} = \frac{1}{0.38} = 0.72 \]

\textbf{Step 11:} Calculating the ripple effect of the indexpage webpage, we have to follow the following rule:

\[ RE_{\text{indexpage}} = \frac{RE_{\text{aspx}} + RE_{\text{aspx.vb}}}{2} \]
\[ = \frac{0 + 0.38}{2} = 0.19 \]
\[ LS_{\text{indexpage}} = \frac{1}{0.19} = 0.84 \]

\textbf{5.2 Computing Ripple Effect for the “Personal” page}

Before we move in to calculating the Ripple Effect of the “personal” page, we have to study the structure of this page first, so that to know how we are going to move on in the process as discussed in chapter 4. Looking at the code (Fig 5.2.1, Fig 5.2.2) of the Personal page (Fig 5.2) we can notice that there are two main files “personal.aspx” and “personal.aspx.vb” so this page uses a code behind file to work. In the next section we will calculate the ripple effect of each page alone then calculating the total Ripple Effect of the “personal” page at the end of section 5.2.2

![Lebanese American University](image)

To check your records click here
Select a Semester to view your schedule

<table>
<thead>
<tr>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall 2006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Name</th>
<th>Room</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Computer Programming I</td>
<td>Sage G01</td>
<td>8:30 - 9:30</td>
</tr>
<tr>
<td>5678</td>
<td>English 101</td>
<td>Niel 110</td>
<td>9:30 - 10:30</td>
</tr>
<tr>
<td>99887</td>
<td>Arabic 101</td>
<td>Niel G015</td>
<td>2:30 - 3:20</td>
</tr>
</tbody>
</table>

Fig 5.2: a user chose to check the Fall 2006 semester schedule, it is displayed in a table
5.2.1 Computing Ripple Effect for the Personal.aspx file

We can find that the personal.aspx presentation file contains no scripts and only one form.

```xml
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="Personal.aspx.vb" Inherits="TestLogin.Personal" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Student Schedule Page</title>
</head>
<body MS_POSITIONING="GridLayout" aLink="#33cc33">
<form method="post" runat="server">
  <asp:DropDownList ID="Sem1" style="Z-INDEX: 103; LEFT: 136px; POSITION: absolute; TOP: 224px; runat="server" Width="168px" Height="40px">
    <asp:ListItem></asp:ListItem>
    <asp:ListItem Value="Fall2005">Fall2005</asp:ListItem>
    <asp:ListItem Value="Spring2006">Spring2006</asp:ListItem>
  </asp:DropDownList>
  <asp:Label ID="Label1" style="Z-INDEX: 104; LEFT: 24px; POSITION: absolute; TOP: 168px; runat="server" Font-Bold="True" ForeColor="#004000" Width="96px" Height="32px">Semester</asp:Label>
  <asp:Label ID="Label1" style="Z-INDEX: 105; LEFT: 24px; POSITION: absolute; TOP: 168px; runat="server" Font-Bold="True" ForeColor="#004000" Width="304px" Height="24px">Select a Semester to view you schedule</asp:Label>
</form>
```

81
Fig 5.2.1: Code written in ASP/HTML for personal.aspx presentation file

The computation of the ripple effect of personal.aspx should proceed as follows:

In chapter 3 we concluded a rule that is: *the ripple effect of any .aspx file that contains one form and no scripts* (i.e. following the first way in creating .aspx file) *is always zero and its logical stability is always 1.*
The Ripple Effect is,
\[ \text{RE}_\text{personal.aspx} = 0 \]
The logical stability is,
\[ \text{LS}_\text{personal.aspx} = 1/(1 + \text{RE}_\text{personal.aspx}) = 1 \]

5.2.2 Computing Ripple Effect for the Personal.aspx.vb file

We can find that the personal.aspx.vb is made from one class called `personal` and this class contains two methods, the first one is loaded automatically when the webpage is loaded called “page_load” and the second one is called when the user clicks on the “check” button in the `personal` page called “check_click”. Fig 5.2.2 represents the code behind file called “personal.aspx.vb”:

```vbnet
Imports TestLogin.indexpage
Imports System.Data.OleDb
Public Class Personal
    Inherits System.Web.UI.Page
    Public cn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
        Data Source=C:\Inetpub\wwwroot\TestLogin\test.mdb;")
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim cmd As OleDbCommand
        Dim dr As OleDbDataReader
        Dim cmd2 As OleDbCommand
        cn.Open()
        cmd = New OleDbCommand("select * from Personal where ID = " &
            TestLogin.indexpage.user_ID & ",", cn)
        dr = cmd.ExecuteReader
        dr.Read()
        lbl_welcome.Text = "Welcome " & dr(2) & dr(3)
        dr.Close()
        cn.Close()
    End Sub

    Private Sub Check_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Check.Click
        Dim cmd_semester As OleDb.OleDbCommand
        Dim schedule As String
        cn.Open()
        schedule = "SELECT CRN,CRTitle,CRBldg,CRTime FROM Course where Semester= ", & Sem.SelectedValue & " and UserID= ", &
            TestLogin.indexpage.user_ID & " "
        cmd_semester = New OleDb.OleDbCommand(schedule, cn)
        Course_Grid.DataSource =
    
```
83
5.2.2.1 Computing Ripple Effect for the "page_load" method

Computing the ripple effect of this method is made in several steps, they are as follows:

**Step1**: make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance

- Cn.open, cmd, TestLogin.indexpage.user_ID, cn, dr, cmd.execute, dr.read,
  - lbl_welcome.text, dr(2), dr(3), dr.close, cn.close
  - Cn.open will be presented as cn.o
  - TestLogin.indexpage.user_ID will be presented as t_user
  - cmd.execute will be presented as cmd.x
  - dr.read will be presented as dr.r
  - lbl_welcome.text will be presented as lbl
  - dr.close will be presented as dr.c
  - cn.close will be presented as cn.c
  - dr in line 12 will be presented as dr_d because its defined
  - cn in line 11 will be presented as cn_a because it is used in defining cmd

**Step2**: Form a Boolean matrix for each method.

\[
V_{load} =\begin{pmatrix}
cno & cmd & tuser & cn_a & dr_d & cmdx & drr & lbl & dr2 & dr3 & drc & cn_c \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]
**Step 3:** Construct a matrix $S$ for `page_load` method

$$S_{load} = \begin{pmatrix}
\text{cno} & \text{cmd} & \text{tuser} & \text{cn} & \text{dr} & \text{cmdx} & \text{drr} & \text{lbl} & \text{dr2} & \text{dr3} & \text{drc} & \text{cnc} \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

**Step 4:** Construct a matrix $X$ for `page_load` method

$$X_{load} = \begin{pmatrix}
\text{cno} & \text{cmd} & \text{tuser} & \text{cn} & \text{dr} & \text{cmdx} & \text{drr} & \text{lbl} & \text{dr2} & \text{dr3} & \text{drc} & \text{cnc} \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix}$$
**Step 5:** Find the Boolean product of $S$ and $X$

\[ S_{load} \times X_{load} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \]

**Step 6:** Find the Boolean product of $V$ with $S$ and $X$

\[ V_{load} \times S_{load} \times X_{load} = (1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1) \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} = (0 \ 2) \]
Step 7: Construct the complexity matrix $C$

$$C = \begin{pmatrix} P_{load} & 2 \\ Check\_click & 3 \end{pmatrix}$$

Note that the "&" in the select statements should be counted in the complexity measurement.

Step 8: Find the product matrix of $C$ with $V, S$ and $X$

$$V_{load} \cdot S_{load} \cdot X_{load} \cdot C = \begin{pmatrix} 2 \\ 0 & 2 \\ 3 \end{pmatrix} = 6$$

Step 9: Divide this figure by $\sum_{i=1}^{r} q_{i} \delta_{i}$ to get the $RE$ and find the reciprocal of $RE$ to get the logical stability.

$$RE_{load} = (V_{load} \cdot S_{load} \cdot X_{load} \cdot C) / \sum_{i=1}^{r} q_{i} \delta_{i} = 6/57 = 0.1$$

The logical stability measure for page_load is defined to be the reciprocal of the ripple effect value. In our code the $RE$ was 0.27 so the logical stability will be:

$$LS_{load} = 1/(RE_{load} + 1) = 1/1.1 = 0.9$$

5.2.2.2 Computing Ripple Effect for the "check_click" method

Computing the ripple effect of this method is made in several steps, they are as follows:

Step 1: make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance
Cn.open, schedule, sem, TestLogin.indexpage.user_ID, cmd_semester, schedule, 
cn, course_grid.ds, cmd_semester.execute, course_grid.db.cn.close

Cn.open will be presented as cn.o
TestLogin.indexpage.user_ID will be presented as t_user
cmd.execute will be presented as cmd.x
cn.close will be presented as cn.c
cmd_semester will be presented as cmds
cmd_semester.execute will be presented as cmdsx
coursegrid.ds will be presented as cgds
coursegrid.db will be presented as cgdb
schedule will be presented as sch
schedule in line 23 will be presented as sch_a because its used
CN in line 23 will be presented as CN_a because it is used

**Step 2:** Form a Boolean matrix for each method.

\[
V_{\text{check}} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

**Step 3:** Construct a matrix \( S_{\text{check}} \) for `check_click` method

\[
S_{\text{check}} = \begin{pmatrix}
cno & sch & sem & tuser & cmds & sch_a & cn_a & cgds & cmdsx & cgdb & cnc
\end{pmatrix}
\]

\[
S_{\text{check}} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
Step 4: Construct a matrix $X$ for check_click method

$$X_{\text{check}} = \begin{pmatrix}
\text{cno} & 1 & 0 \\
\text{sch} & 0 & 0 \\
\text{sem} & 0 & 0 \\
\text{tuser} & 0 & 0 \\
\text{cmds} & 0 & 0 \\
\text{schu} & 0 & 0 \\
\text{cnu} & 0 & 0 \\
\text{cgds} & 0 & 0 \\
\text{cmdx} & 0 & 0 \\
\text{cgdb} & 0 & 0 \\
\text{cnc} & 1 & 0 \\
\end{pmatrix}$$

Step 5: Find the Boolean product of $S$ and $X$

$$S_{\text{check}} X_{\text{check}} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \begin{pmatrix}
1 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
\end{pmatrix} = \begin{pmatrix}
2 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
\end{pmatrix}$$

89
**Step 6:** Find the Boolean product of \( V \) with \( S \) and \( X \)

\[
V_{\text{check}} S_{\text{check}} X_{\text{check}} = \begin{pmatrix}
2 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0
\end{pmatrix}
= \begin{pmatrix}
2 \\
0
\end{pmatrix}
\]

**Step 7:** Construct the complexity matrix \( C \)

\[
C = \begin{pmatrix}
P_{\text{load}} \\
\text{Check}_\text{click}
\end{pmatrix}
\begin{pmatrix}
2 \\
3
\end{pmatrix}
\]

Note that the "&" in the select statements should be counted in the complexity measurement.

**Step 8:** Find the product matrix of \( C \) with \( V, S \) and \( X \)

\[
V_{\text{check}} S_{\text{check}} X_{\text{check}} C = \begin{pmatrix}
2 & 0 \\
3 & 0
\end{pmatrix}
= 4
\]
**Step 9:** Divide this figure by $\sum_{i=1}^{\Omega} \theta_i \delta_i$ to get the $RE$ and find the reciprocal of $RE$ to get the logical stability.

$$RE_{\text{check}} = \frac{(V_{\text{check}} \cdot S_{\text{check}} \cdot X_{\text{check}} \cdot C)}{\sum_{i=1}^{\Omega} \theta_i \delta_i} = \frac{4}{57} = 0.07$$

The logical stability measure for check_click is defined to be the reciprocal of the ripple effect value. In our code the $RE$ was 0.22 so the logical stability will be:

$$LS_{\text{check}} = \frac{1}{(RE_{\text{check}} + 1)} = \frac{1}{1.07} = 0.93$$

**Step 10:** Calculating the ripple effect of the personal.aspx.vb file, we have to follow the following rule:

$$\frac{1}{n} \sum_{i=1}^{n} \frac{(V_{m_i} \cdot S_{m_i} \cdot X_{m_i} \cdot C)}{\sum_{i=1}^{\Omega} \theta_i \delta_i}$$

Where $m$ is a method and $n$ is the number of total methods in the .aspx.vb file.

$$RE_{\text{personal.aspx.vb}} = (0.1+0.07)/2 = 0.08$$

$$LS_{\text{personal.aspx.vb}} = 1 / 1.08 = 0.92$$

**Step 11:** Calculating the ripple effect of the personal webpage, we have to follow the following rule:

$$RE_{\text{personal}} = \frac{(RE_{\text{personal.aspx}} + RE_{\text{personal.aspx.vb}})}{2}$$

$$= (0 + 0.08)/2 = 0.04$$

$$LS_{\text{personal}} = 1 / 1.04 = 0.96$$

**5.3 Computing Ripple Effect for the “Grades” page**

In order to move on in calculating the Ripple Effect of the “grades” page (Fig 5.3 (b) ), we have to study the structure of this page first, so that to know how we are going to move on in the process as discussed in chapter 4. Looking at the code of the grades page we can notice that there is no code behind, in fact the code behind code was moved into a script on purpose so that we will give an example in calculating the ripple effect of
an ASP.net page that is made from only one file “grades.aspx” (Fig 5.3 (a) ) the code in
the script follows the Visual Basic.net language as that of the codebehind. In the next
section we calculate the ripple effect of each section alone (ASP/HTML code then the
script) then we calculate the total Ripple Effect of the “grades” page at the end of section
5.3.2. Note that in section 5.4 we will discuss the different results that we obtained
clarifying the role that Ripple Effect measurement it can play in Software quality
assurance and maintenance.

<script language = VB runat= server>
1 Imports TestLogin.indexpage
2 Imports System.Data.OleDb
3 Public Class grades
4 Inherits System.Web.UI.Page
5 Public cn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
 Data Source=C:\Inetpub\wwwroot\TestLogin\test.mdb;")
6 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
7 Dim cmd As OleDbCommand
8 Dim dr As OleDbDataReader
9 Dim cmd2 As OleDbCommand
10 cn.Open()
11 cmd = New OleDbCommand("select * from Personal where ID = "; &
 TestLogin.indexpage.user_ID & ",", cn)
12 dr = cmd.ExecuteReader
13 dr.Read()
14 lbl_welcome.Text = "Welcome " & dr(2) & dr(3)
15 dr.Close()
16 cn.Close()
18 End Sub
19
20 Private Sub Check_Click()
21 Dim cmd_semester As OleDb.OleDbCommand
22 Dim Record_grade As String
23 cn.Open()
24 Record_grade = "SELECT CRN,CRTitle,CRBldg,CRTime,Grade FROM
 Record where Semester= '" & Sem.SelectedValue & '" and UserID= "; & TestLogin.indexpage.user_ID & ","
25 cmd_semester = New OleDb.OleDbCommand(Record_grade, cn)
26 Record_Grid.DataSource =
27 cmd_semester.ExecuteReader(CommandBehavior.SequentialAccess)
28 Record_Grid.DataBind()
29 cn.Close()
30 End Sub
32 End Class
</script>

<HTML>
<HEAD>

92
Fig 5.3 (a): Code written in VB.Net for the script in grades.aspx.vb
Fig 5.3 (b): a user chose to check the Fall 2006 semester grades, it is displayed in a table.

5.3.1 Computing Ripple Effect for the Script in grades.aspx

The script written in grades.aspx (Fig 5.3 (a)) is made from one class called grades and two methods named “page_load” and “check_click”. In order to calculate the ripple effect of the script as a whole we have to calculate the RE of the two methods separately then coming up with the total RE in section 5.3.1.2.

5.3.1.1 Computing Ripple Effect for the “page_load” method

Computing the ripple effect of this method is made in several steps, they are as follows:

**Step 1:** make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance

```
Cn.open, cmd, TestLogin.indexpage.user_ID, cn, dr, cmd.execute, dr.read,
lbl_welcome.text, dr(2), dr(3), dr.close, cn.close
```

Cn.open will be presented as cn.o
TestLogin.indexpage.user_ID will be presented as t_user
cmd.execute will be presented as cmd.x
dr.read will be presented as dr.r
lbl_welcome.text will be presented as lbl
dr.close will be presented as dr.c
cn.close will be presented as cn.c
dr in line 13 will be presented as dr_d because its defined
cn in line 12 will be presented as cn_n because it is used in defining cmd

**Step2:** Form a Boolean matrix for each method.

\[
V_{load} = \begin{pmatrix}
cno & cmd & tuser & cn_n & dr_d & cmox & drr & lbl & dr2 & dr3 & drc & cnc \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{pmatrix}
\]

**Step3:** Construct a matrix \( S \) for `page_load` method

\[
S_{load} = \begin{pmatrix}
cno & cmd & cn_n & dr_d & cmox & drr & lbl & dr2 & dr3 & drc & cnc \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
cmd & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
tuser & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
cn_n & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
dr_d & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
cmox & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
drr & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
lbl & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
\]
**Step 4:** Construct a matrix $X$ for page load method

$$
X_{\text{load}} = \begin{pmatrix}
\text{cno} & 0 & 1 & 1 \\
\text{cmd} & 0 & 0 & 0 \\
\text{tuser} & 0 & 0 & 0 \\
\text{cnr} & 0 & 0 & 0 \\
\text{dr_d} & 0 & 0 & 0 \\
\text{cmdx} & 0 & 0 & 0 \\
\text{drr} & 0 & 0 & 0 \\
\text{lbl} & 0 & 0 & 0 \\
\text{dr2} & 0 & 0 & 1 \\
\text{dr3} & 0 & 0 & 1 \\
\text{drc} & 0 & 0 & 0 \\
\text{cnc} & 0 & 0 & 0 \\
\end{pmatrix}
$$

**Step 5:** Find the Boolean product of $S$ and $X$

$$
S_{\text{load}} X_{\text{load}} = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
0 & 1 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
= \begin{pmatrix}
0 & 1 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
$$
**Step 6:** Find the Boolean product of $V$ with $S$ and $X$

$$V_{load} \cdot S_{load} \cdot X_{load} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 7 \end{pmatrix}$$

**Step 7:** Construct the complexity matrix $C$

$$C = \begin{pmatrix} P_{load} \\ Check\_click \\ Form1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

*Note that the "&" in the select statements should be counted in the complexity measurement.*

**Step 8:** Find the product matrix of $C$ with $V$, $S$ and $X$

$$V_{load} \cdot S_{load} \cdot X_{load} \cdot C = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = 31$$
**Step 9:** Divide this figure by $\sum \theta_i \delta_i$ to get the RE and find the reciprocal of RE to get the logical stability.

$$RE_{load} = \frac{(V_{load} \cdot S_{load} \cdot C_{load} \cdot C)}{\sum \theta_i \delta_i} = 31/89 = 0.34$$

The logical stability measure for page_load is defined to be the reciprocal of the ripple effect value. In our code the RE was 0.27 so the logical stability will be:

$$LS_{load} = 1/(RE_{load} + 1) = 1/1.34 = 0.74$$

### 5.3.1.2 Computing Ripple Effect for the “check_click” method

Computing the ripple effect of this method is made in several steps, they are as follows:

**Step 1:** make a list of all occurrences of variables (not including the declarations) for each method according to the order or appearance

- Cn.open, record_grade, sem, TestLogin.indexpage.user_ID, cmd_semester, record_grade, cn, record_grid.ds, cmd_semester.execute, course_grid.db, cn.close

  Cn.open will be presented as cn.o

  TestLogin.indexpage.user_ID will be presented as t_user

  cmd.execute will be presented as cmd.x

  cn.close will be presented as cn.c

  cmd_semester will be presented as cmds

  cmd_semester.execute will be presented as cmdsx

  record_grid.ds will be presented as rgds

  record_grid.db will be presented as rgdb

  record_grade in line 22 will be presented as rcd_d

  record_grade in line 23 will be presented as rcd_u because its used

  cn in line 23 will be presented as cn_u because it is used
**Step 2:** Form a Boolean matrix for each method.

$$V_{check} = \begin{bmatrix}
\text{cno} & \text{rcd}_d & \text{sem} & \text{tuser} & \text{cmds} & \text{rcd}_a & \text{cn}_a & \text{rgds} & \text{cmdx} & \text{rgdb} & \text{cn}_c \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0
\end{bmatrix}$$

**Step 3:** Construct a matrix $S$ for `check_click` method

$$S_{check} = \begin{bmatrix}
\text{cno} & \text{rcd}_d & \text{sem} & \text{tuser} & \text{cmds} & \text{rcd}_a & \text{cn}_a & \text{rgds} & \text{cmdx} & \text{rgdb} & \text{cn}_c \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

**Step 4:** Construct a matrix $X$ for `check_click` method

$$X_{check} = \begin{bmatrix}
\text{cno} & \text{rcd}_d & \text{sem} & \text{tuser} & \text{cmds} & \text{rcd}_a & \text{cn}_a & \text{rgds} & \text{cmdx} & \text{rgdb} & \text{cn}_c \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$
**Step 5:** Find the Boolean product of $S$ and $X$

\[
S_{\text{check}} \times X_{\text{check}} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 2 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 2 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\]

**Step 6:** Find the Boolean product of $V$ with $S$ and $X$

\[
V_{\text{check}} \times S_{\text{check}} \times X_{\text{check}} = (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0)
\]

\[
\begin{pmatrix}
1 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 2 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} = \begin{pmatrix}
1 & 0 & 4
\end{pmatrix}
\]
**Step 7:** Construct the complexity matrix C

\[
C = \begin{pmatrix}
  P_{\text{load}} & 2 \\
  \text{Check click} & 3 \\
  \text{Form1} & 4
\end{pmatrix}
\]

*Note that the "&" in the select statements should be counted in the complexity measurement.*

**Step 8:** Find the product matrix of C with \(V, S\) and \(X\)

\[
V_{\text{check}} \cdot S_{\text{check}} \cdot X_{\text{check}} \cdot C = \begin{pmatrix}
  2 \\
  3 \\
  4
\end{pmatrix}
\begin{pmatrix}
  1 & 0 & 4 \\
  \end{pmatrix} = 18
\]

**Step 9:** Divide this figure by \(\sum \theta_i \delta_i\) to get the RE and find the reciprocal of \(RE\) to get the logical stability.

\[
RE_{\text{check}} = (V_{\text{check}} \cdot S_{\text{check}} \cdot X_{\text{check}} \cdot C) / \sum \theta_i \delta_i = 18/89 = 0.2
\]

The logical stability measure for \(\text{check click}\) is defined to be the reciprocal of the ripple effect value. In our code the \(RE\) was 0.22 so the logical stability will be:

\[
LS_{\text{check}} = 1/(RE_{\text{check}} + 1) = 1/1.2 = 0.83
\]

### 5.3.2 Computing Ripple Effect for the ASP/HTML in grades.aspx

Computing the Asp/Html code in grades.aspx is the same as computing a normal .aspx file, but now we have a script embedded. So, a lot of difference will be observed in
creating the \( X \) matrix and thus resulting in a \( RE \) number instead of zero as we found in "indexpage.aspx" and "personal.aspx".

To start in calculating the Ripple Effect of ASP/HTML code in grades.aspx we simply follow the usual steps:

**Step 1:** make a list of all occurrences of controls (not including labels) for each Form, the order of appearance is not important

- \( \text{lbl} \) represents \( \text{lbl}_{\text{welcome}} \)
- \( \text{h2} \) represents \( \text{hyperlink2} \)
- \( \text{rg} \) represents \( \text{record\_grid} \)
- \( \text{chk} \) represents \( \text{check\_button} \)
- \( \text{sv} \) represents \( \text{sem\_field\_validator} \)
- \( \text{l1} \) represents the first item in the list
- \( \text{l2} \) represents the second item in the list

**Step 2:** Form a Boolean matrix for each form.

\[
V_{F1} = \begin{bmatrix}
\text{Llbl} & \text{h2} & \text{rg} & \text{chk} & \text{sv} & \text{l1} & \text{l2} & \text{sem} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

**Step 3:** Construct a matrix \( S \) for \textit{Form1}

\[
S_{F1} = \begin{bmatrix}
\text{lbl} & \text{h2} & \text{rg} & \text{chk} & \text{sv} & \text{l1} & \text{l2} & \text{sem} \\
\text{lbl} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{h2} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{rg} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{chk} & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\text{sv} & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\text{l1} & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
\text{l2} & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\text{sem} & 0 & 0 & 1 & 0 & 0 & 1 & 1
\end{bmatrix}
\]
**Step 4:** Construct a matrix $X$ for `check_click` method

$$
\begin{align*}
P & \begin{pmatrix}
\text{load} & \text{Check} & \text{Form1} \\
\text{lbl} & 1 & 0 & 0 \\
\text{h2} & 0 & 0 & 0 \\
\text{rg} & 0 & 1 & 0 \\
\text{chk} & 0 & 1 & 0 \\
\text{sv} & 0 & 0 & 0 \\
\text{l1} & 0 & 1 & 0 \\
\text{l2} & 0 & 1 & 0 \\
\text{sem} & 0 & 1 & 1
\end{pmatrix}
\end{align*}
$$

**Step 5:** Find the Boolean product of $S$ and $X$

$$
S_{F1} X_{F1} = 
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 1
\end{pmatrix} = 
\begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 2 & 0 \\
0 & 2 & 1 \\
0 & 4 & 1
\end{pmatrix}
$$

**Step 6:** Find the Boolean product of $V$ with $S$ and $X$

$$
V_{F1} S_{F1} X_{F1} = (1 \ 1 \ 1 \ 1 \ 1 \ 1) \begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 2 & 0 \\
0 & 1 & 0 \\
0 & 2 & 1 \\
0 & 4 & 1
\end{pmatrix} = (1 \ 12 \ 3)
$$
Step 7: Construct the complexity matrix $C$

$$C = \begin{pmatrix} P_{\text{load}} & 2 \\ \text{Check\_click} & 3 \\ \text{Form1} & 4 \end{pmatrix}$$

Note that the "&" in the select statements should be counted in the complexity measurement.

Step 8: Find the product matrix of $C$ with $V$, $S$ and $X$

$$V_{\text{check}} \cdot S_{\text{check}} \cdot X_{\text{check}} \cdot C = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = 50$$

Step 9: Divide this figure by $\sum_{i=1}^{n} \theta_i \delta_i$ to get the $RE$ and find the reciprocal of $RE$ to get the logical stability.

$$RE_{F1} = \frac{V_{F1} \cdot S_{F1} \cdot X_{F1} \cdot C}{\sum_{i=1}^{n} \theta_i \delta_i} = \frac{50}{89} = 0.56$$

The logical stability measure for Form1 is defined to be the reciprocal of the ripple effect value. In our code the $RE$ was 6.25 so the logical stability will be:

$$LS_{F1} = \frac{1}{RE_{F1} + 1} = \frac{1}{1.56} = 0.64$$

Step 10: Calculating the ripple effect of the whole grades webpage, we have to follow the following rule:

$$\frac{1}{l} \sum_{i=1}^{l} \frac{(V_{ki} \cdot S_{ki} \cdot X_{ki} \cdot C)}{\sum_{i=1}^{n} \theta_i \delta_i}$$

104
Where “k = a method or a form” and “l = number of total methods and forms” in the .aspx file.

\[ RE_{grades} = \frac{0.34 + 0.2 + 0.56}{3} = 0.36 \]

\[ LS_{grades} = 1 / 1.36 = 0.73 \]

5.4 Computing Ripple Effect for the website as a whole

After finding the ripple effect of each page in the “Testlogin” website. We have to find the ripple effect and logical stability of the whole website which is calculated by:

\[ RE_{Testlogin} = \frac{1}{w} \sum_{i=1}^{w} RE_{pi} \]

Where \( w \) is the number of pages in a website and \( p \) is a single web page.

According to the above rule, the web pages’ ripple effects were:

\[ RE_{indexpage} = 0.26 \]

\[ RE_{personal} = 0.04 \]

\[ RE_{grades} = 0.36 \]

Then the Ripple effect of the whole website will be:

\[ RE_{Testlogin} = \frac{0.26 + 0.04 + 0.36}{3} = 0.22 \]

The logical stability will therefore be: \( LS_{Testlogin} = \frac{1}{1.22} = 0.82 \)

These numbers of LS and RE should be read as relative and not absolute, so that to compare the RE and LS of different websites.
5.5 Results and Analysis

Table 5.5 illustrates some interesting facts in developing a high quality and a stable .Net website.

Table 5.5: testlogin Ripple Effect and Logical Stability results

<table>
<thead>
<tr>
<th></th>
<th>Ripple Effect</th>
<th>Logical Stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexpage</td>
<td>0.19</td>
<td>0.84</td>
</tr>
<tr>
<td>Personal</td>
<td>0.04</td>
<td>0.96</td>
</tr>
<tr>
<td>Grades</td>
<td>0.36</td>
<td>0.73</td>
</tr>
<tr>
<td>Total</td>
<td>0.59</td>
<td>0.84</td>
</tr>
</tbody>
</table>

In Table 5.5 the results of computing the RE and LS of every web page in TestLogin website are illustrated. Note that “indexpage” and “Personal” pages follow the same way in creating a webpage (pure ASP/HTML in .aspx file and code behind) while the “grades” page follows a different way where it never uses a codebehind file instead it uses a script inside the .aspx file. As ripple effect increases the logical stability of a program decreases, thus the webpage having higher ripple effect will face more problems when updated and they are the least stable. The results in the table shows that the “grades” webpage have the highest Ripple effect, thus the lowest Logical stability among all the webpages forming the TestLogin website. In my research I applied more examples and I came up with the same conclusion:

Webpages that are developed using Code Behind as the main file in establishing the functionality of a website have the lowest Ripple Effect and the highest Logical Stability measurement. And that is what Microsoft’s target was, to have a stable website while working with the code behind. Thus the approach we worked with and the result we came up with are accurate and exact. On the contrary webpages that are developed not using the code behind file at all will face instability while updating them because they have the highest Ripple Effect measurement and the lowest Logical Stability.
Chapter 6

Conclusions and Further work

In this thesis, our motivation was to compute the ripple effect and logical stability for web applications focusing on .Net environment (ASP.Net and VB.Net) using matrix arithmetic. We used a static approach on code level to do the work. In order to calculate the ripple effect of a .Net web application we studied all .Net dependencies (ASP.Net and OOP VB.Net), relations and propagations. We cleared the idea about the three ways in creating an ASP.Net web page. We made a further analysis on some previous work in calculating the ripple effect. We manipulated them to make it possible to calculate the Ripple Effect of a website based on the three different ways.

6.1 Change Propagations and complexity measurement

We introduced two new terms in change propagation with their own conditions "Form-Scope Change Propagation" and "Page-Scope Change Propagation". The computed ripple effect is not based only on the amount of propagation derived from ASP.Net and VB.Net dependences, but was also weighted by complexity derived by establishing a new technique in calculating the complexity measurement for a Form written in ASP/HTML code specifically in .aspx file following set of conditions called "BABA's Complexity Measurement". This makes the final ripple effect value more reflective of the cost of different program changes. Our algorithm calculate the ripple effect for .Net web application at the code level by calculating both Form-Scope Change Propagation and Page-Scope Change Propagation for ASP.Net and Inter-class propagation and Intra-class propagation for VB.Net and VBScripts. In addition, each matrix used within the algorithm holds a particular kind of information about the software under analysis. It
makes it easier to understand what each part of the algorithm means and how the ripple effect is being computed. One of the complexities of maintenance is that it is not clear what impact will the local changes enforce on the rest of the code. What can be used to help estimating the impact is the ripple effect that we compute, among other measures and metrics. However, our suggested technique for computing the ripple effect of local changes have limitations. One limitation is the time taken to compute the ripple effect; the indicator is the computation of the propagation matrix (shown in chapter 4 table 4.1). The second limitation is concerned with the program dependences; only "direct" dependences have been considered while semantic and run-time dependences have been ignored.

6.2 Further Work

Previous measures and tools were developed to produce ripple effect measures for procedural software using Yau and Collofello’s algorithm which is based on set theory. Obviously it is difficult to write a simple software using this algorithm. RE tools have either taken an excessive amount of time to produce ripple effect measures of needed some user intervention to make critical decisions about the source code. Sue Black produced a software called REST (Ripple Effect and Stability Tool). Several enhancements need to be made which are mainly concerned with size and parsing of target source code before REST can be used in industry. For our approach, we need to develop a complete parser that will pick up the data from .aspx and .aspx.vb pages and use them in calculating the ripple effect automatically. The parser should be either upgradeable or should be developed to enable .Net upgrades.

Computation of ripple effect for web applications at the design level. If that is possible then the developer can change the complexity and the structure of their code as needed to achieve the lowest Ripple Effect possible with the highest logical stability before starting the programming phase. Thus, developers will not waste their time and effort in developing a fragile code, on the contrary they will be making a solid and an excellent well structured web application.
Computation of ripple effect for distributed systems can be taken into account to be studies in the future, because distributed systems are largely implemented nowadays in medium and large scaled companies.

Computing the ripple effect under different web languages can be possible, that is because each language has its own uniqueness. PHP and J2EE can be deeply discovered, coming up with methods and ways in calculating their Ripple effect. Then it would be extremely important to compare the Ripple Effect and Logical stability results of the same web application functionality with .Net, PHP and J2EE to see what web programming language is more stable and better structured. A tool could be created to calculate the Ripple effect of PHP and J2EE web applications.

Nowadays, mobile phones are getting more and more involved in our everyday activities. Going further into mobile programming, and computing the Ripple Effect of WAP pages/applications or applications already installed on the mobiles created by WML language for example can help stabilizing and maintaining the applications created.

Personal Digital Assistant (PDA) is another kind of technology that is booming. PDA applications on both Pocket PC and Palm OS platforms are being widely used for stock counting, accounting and other uses. It has its unique structure of coding, especially if the applications created works with wireless connectivity with a database server.

Another further work that might take a different target than coding is computing the stability and ripple effect of a database (we have different kinds/structures of databases). Where a change in a field or a relation or a table or query or a macro in the database can affect other fields, relations, tables, queries and macros. So after making sure that a database is well structured and stable we can start creating the software that needs that database to manipulate with its data.
References


