An ASP.NET Web Applications Data Flow Testing Approach

Article *in* International Journal of Computer Applications · November 2016

DOI: 10.5120/ijca2016912117

4 authors, including:

Moheb Girgis
Minia University
68 PUBLICATIONS **1,055** CITATIONS
SEE PROFILE

Alaa Elnashar
Minia University
14 PUBLICATIONS **37** CITATIONS
SEE PROFILE

Tarek Abd El-Hafeez
Minia University
28 PUBLICATIONS **65** CITATIONS
SEE PROFILE

An ASP.NET Web Applications Data Flow Testing Approach

Moheb R. Girgis
Department of Computer Science,
Faculty of Science,
Minia University,
El-Minia, Egypt

Alaa I. El-Nashar
Department of Computer Science,
Faculty of Science,
Minia University,
El-Minia, Egypt

Tarek A. Abd El-Rahman
Department of Computer Science,
Faculty of Science,
Minia University,
El-Minia, Egypt

Marwa A. Mohammed
Department of Computer Science,
Faculty of Science,
Minia University,
El-Minia, Egypt

ABSTRACT
This paper extends data flow testing techniques to Web applications, and presents a proposed approach to data flow testing of ASP.NET Web applications. It discusses the data flow analysis of ASP.NET Web applications, which have different structure than traditional programs. The proposed approach involves the construction of a Web application data flow model to support data flow analysis of ASP.NET Web applications. In this approach, testing is performed in four levels, Function, Interprocdural, Page, and Inter-Page levels. In each level, the definition-use (def-use) chains of the variables of interest are computed, then test data can be generated to cover these def-use chains, in order to fulfill the all-uses criterion. The application of the proposed approach is illustrated through a case study.

General Terms
Software engineering, Software testing, Web applications.

Keywords
Web applications testing, ASP.NET Web applications testing, Data flow testing, Data flow testing techniques, Web applications data flow model.

1. INTRODUCTION
The number of Web applications continues to grow at a rapid pace and the requirements on their performance grow as they become more advanced and are exposed to higher loads of users. This makes the subject of Web application testing important and of current interest. Web application testing is both challenging and critical. It is challenging because traditional testing methods and tools are not sufficient for Web applications. Testing Web applications is critical because failure may be very costly.

The basic structure of Web applications consists of three tiers: the client, the server and the data store. Due to the widespread use of Microsoft's .NET platform, this work focuses on ASP.NET Web applications; nevertheless, the technique presented can be adapted to Web applications developed in various environments. In ASP.NET Web applications, data can be stored in the state/session variables and the ASP.NET server controls, in addition to traditional program variables. So, traditional data flow testing techniques need to be enhanced to manipulate the definition-use associations of such variables.

This paper extends data flow testing techniques to Web applications, and presents a proposed approach to data flow testing of ASP.NET Web applications. It discusses the data flow analysis of ASP.NET Web applications, which have different structure than traditional programs. The proposed approach involves the construction of a Web application data flow model to support data flow analysis of ASP.NET Web applications. In this approach, testing is performed in four levels, ***Function***, ***Interprocdural***, ***Page***, and ***Inter-Page*** levels. In each level, the definition-use (def-use) chains of the variables of interest are computed, then test data can be generated to cover these def-use chains, in order to fulfill the all-uses criterion.

The rest of the paper is organized as follows: Section 2 presents a review of the related work in data flow testing of Web applications. Section 3 describes briefly the idea of data flow analysis, and the data flow testing criterion used in this work. Section 4 describes the proposed Web application data flow model. Section 5 describes the proposed Web applications data flow testing approach. Section 6 presents a case study to illustrate the application of the proposed Web applications data flow testing approach. Section 7 presents the conclusion of the research work presented in this paper.

2. RELATED WORK
Research on Web applications testing has been fairly limited. As this study focuses on the data flow testing of Web applications, this section presents a review of the published research work related to this field.

Liu et al. [1, 2] have proposed an approach for data-flow testing of Web applications. The approach is applicable to Web applications implemented in HTML and XML languages, and including interpreted scripts as well as other kinds of executable components (such as Java applets, ActiveX controls, Java beans, etc.) both at the client and server side of the application. The approach is based on a Web application test model, WATM, which includes an Object model and a Structure model. In the Object model, components are modeled as objects that contain attributes and operations, and the Structural model captures the data flow information of functions within or across objects. Their data-flow testing approach derives test cases from three different perspectives: intra-object, inter-object, inter-client. Five testing levels specifying different scopes of the tests to be run

have been defined, namely: *Function*, *Function Cluster*, *Object*, *Object Cluster*, and *Application level*.

Lee and Offutt [3] have proposed a mutation based approach to test HTML-based Web applications. Their approach focuses on the reliability of the data exchange among Web component via HTML, more specifically, functional correctness of the interactions. Nevertheless, the functionality of individual Web applications is not considered in their work.

Ricca and Tonella [4, 5] have proposed an approach for white box testing of static Web applications. This approach was based on two test models: the Navigational model that focuses on HTML pages and navigational links of the application, and the Control flow model that represents the internal structure of Web pages in terms of the execution flow followed. This latter model has been used to carry out structural testing too. A test case for a Web application is defined as a sequence of pages to be visited, plus the input values to be provided to pages containing forms. Various control flow and data flow coverage criteria applicable on both models have been proposed to design test cases.

Mansour and Houri [6] have presented white box techniques for testing Web applications developed in the .NET environment. These techniques emphasize the distinct features of Web-based programs, including their multi-tier nature, extensive use of events, and hyperlinked structure. First, they extended previous work on modeling Web applications by enhancing previous dependence graphs and proposing an event-based dependence graph model. Second, they applied data flow testing methods to the dependence graphs and proposed an event flow testing technique. Third, they presented a few coverage testing approaches. Fourth, they proposed mutation testing operators for evaluating the adequacy of Web application tests.

Qi et al. [7] have presented an agent-based approach to perform data flow testing of Web applications. In this approach, the data flow testing is performed by autonomous test agents at the method level, object level, and object cluster level. In the process of the recommended data flow testing, an agent-based Web application testing system automatically generates and coordinates test agents to decompose the task of testing an entire Web application into a set of subtasks that can be accomplished by test agents. The test agents cooperate with each other to complete the testing of a Web application.

Liu [8] has adapted traditional data flow testing techniques into the context of Java Server Pages (JSP). The author pointed out that the JSP implicit objects and action tags can introduce several unique data flow test artifacts which need to be addressed. A test model was presented to capture the data flow information of JSP pages with considerations of various implicit objects and action tags. Based on the test model, Liu described an approach to compute the intraprocedural, interprocedural, and sessional data flow test paths for uncovering the data anomalies of JSP pages.

3. DATA FLOW ANALYSIS

Data flow analysis focuses on the interactions between variable definitions (defs) and references (uses) in a program, i.e. the def-use chains. Data flow analysis techniques use a control flow graph representation of a program to compute def-use chains.

The control flow of a program can be represented by a directed graph, with a set of nodes and a set of edges, called the control flow graph (CFG). Each node represents a statement. The edges of the graph are possible transfers of control flow between the nodes. A path is a finite sequence of nodes connected by edges. A complete path is a path whose first node is the start node and whose last node is an exit node. A path is def-clear with respect to a variable if it contains no new definition of that variable.

To determine whether the testing process is finished, a test data adequacy criterion is needed [9]. Several test data adequacy criteria have been proposed, such as control flow-based and data flow-based criteria. The proposed data flow testing approach is based on one of the data flow testing criteria proposed by Rapps and Weyuker [10], which is the ***all-uses criterion***. It requires a def-clear path from each def of a variable to each use of that variable to be traversed. The def-clear paths required to satisfy the all-uses criterion, called ***du-paths***, are constructed from the defs and uses of program variables by using the technique described in [11].

4. THE PROPOSED WEB APPLICATION DATA FLOW MODEL

The first step in the proposed Web applications data flow testing approach is to capture data flow information about the Web application to be tested, i.e. perform data flow analysis of the Web application. The first task in data flow analysis is to build a data flow model for the Web application to be analyzed. The proposed Web application data flow model consists of four types of flow graphs: control flow graph (CFG), interprocedural control flow graph (ICFG), page control flow graph (PCFG), and composite control flow graph (CCFG). These flow graphs are described below.

4.1 Control Flow Graph

The *control flow graph* (CFG) is used to depict the data flow information of an individual function. To describe data flow information, a CFG is annotated with the defs/uses of the variables so that the def-use chains for the variables of interest can be obtained from it. For example, a def-use chain <(v, i), j)> exists if there is a *du-path* from the def of variable v at node i to its use at node j.

4.2 Interprocedural Control Flow Graph

The *interprocedural control flow* graph (ICFG) is used to describe the data flow information that involves more than one function. An ICFG integrates the CFGs of calling and called functions into a single entry, single exit CFG. Thus, the def-use chains for a variable that is defined in one function and used in other functions can be obtained from the ICFG.

4.3 Page Control Flow Graph

An ASP.NET Web page consists mainly of a 'presentation file' (.aspx file), that contains server controls and XHTML tags, and the 'code-behind' class (.aspx.cs file) that contains functions and data items required for the class to perform specific actions. The *page control flow* graph (PCFG) is used to describe the data flow information between the ASPX file and the code-behind class of a page. A PCFG for a page integrates the CFG of the ASPX file and ICFG of the related code-behind class into a single entry, single exit CFG. Thus, the def-use chains for a variable that is defined in one part of the page and used in the other part can be obtained from the PCFG.

4.4 Composite Control Flow Graph

One of the important characteristics of Web applications is that the data in a Web page can be passed to another page when users click a hyperlink or submit a form, or programmatically through Response.Redirect() and

Server.Transfer() methods. Thus, a variable can be defined in one page and used in another page. However, unlike traditional programs, there are no direct calling relations between the embedded functions across the Web pages and, hence, the ICFG cannot be used to capture this kind of data interactions. Therefore, to describe the data flow information between interacting Web pages, the *composite control flow* graph (CCFG) is introduced so that the def-use chains across Web pages can be obtained, [1].

The CCFG can be constructed by connecting the related PCFGs of the interacting Web pages together. For each PCFG of a page, there exists an edge from its exit node to the entry node of another page's PCFG. Thus, with the CCFG, the def-use chains of interacting Web pages can be obtained.

5. THE PROPOSED WEB APPLICATIONS DATA FLOW TESTING APPROACH

This section describes the proposed Web applications data flow testing approach. In this approach, data flow analysis of the Web application to be tested is performed first, to collect information about the defs and uses of its variables. Then, using this information, with the constructed Web application data flow model, and the data flow testing technique described in [11], the def-use chains of the Web application variables are computed. Because of the distinctive structure of Web applications, in the proposed data flow testing approach, testing is performed in four levels, Function, Interprocdural, Page, and Inter-Page levels. In each level, the def-use chains are computed, and test data are generated to cover all of them, if possible.

The following sub-sections describe the data flow analysis of ASP.NET pages, and the four testing levels.

5.1 Data Flow Analysis of ASP.NET Pages

Data flow analysis is mainly concerned with the proper usage of data in a program. In traditional programs, data are stored in the program variables. In Web applications, data can be stored in the state/session variables and the ASP.NET server controls, in addition to traditional program variables. Therefore, data flow analysis techniques need to be extended to address the distinguishing data elements of ASP.NET pages, which are the basic building blocks of any Web application. This subsection describes the issues that characterize data flow analysis of ASP.NET pages.

To test ASP.NET pages, not only the variables of the ASP.NET pages need to be considered, but also the implicit objects introduced by the ASP.NET technology. So, in the proposed data flow testing approach, the definitions and uses of five types of data objects that can be found in Web applications are considered:

1. Traditional program variables and arrays
2. Instance variables of the code-behind class
3. Simple and complex ASP.NET server controls and their properties
4. Implicit session/state objects, such as Query-string parameters, ViewState property, and Session property.
5. Objects of built-in classes, such as SQL server classes, ADO.NET classes, and custom button controls.

The def and use actions for traditional program variables and arrays are determined as described in [11]. The def and use actions for the other four types of the ASP.NET pages data objects are described below.

5.1.1 Defs and uses of instance variables of the code-behind class

In object-oriented (OO) programming the instance variables of a class can be accessed and modified by its methods. In ASP.NET pages, the instance variables of the code-behind class can not only be accessed and modified by its methods, but also referenced in the related ASPX file.

The defs and uses of the instance variables of the code-behind class are defined as in OO programs. But, to illustrate how the instance variables of the code-behind class of a page can be defined in one of its methods and used in the related ASPX file, consider the following code fragments of a page, named Welcome.aspx.

```
0    <%@ Page Language="C#" AutoEventWireup
     ="true" CodeFile="Welcome.aspx.cs" Inherits
     ="Welcome" %>
     …
5    <title>
6      <%=title%>
7    </title>
     …
50   public partial class Welcome : System.Web.UI.Page
51   {
52     protected string title;
53     public string str;
       …
60     protected void Page_Load (object sender,
                                        EventArgs e)
61     {
         …
65       str = Request.Params["id"];
66       title = "Welcome" + str;
         …
80     }
```

In this example, the code-behind class, Welcome.aspx.cs, has two instance variables, str and title. In the Page_Load method, there are ***def*** and ***use*** actions for str at lines 65 and 66, respectively. The variable title has a ***def*** at line 66 in the Page_Load method, and a ***use*** at line 5 (an output statement) in the ASPX file.

5.1.2 Defs and uses of server controls and their properties

The ASP.NET server controls in the ASPX file are treated as global variables, which can be accessed (defined/used) by the code in the code-behind file. Thus, to apply data flow testing to Web applications, it is necessary to determine first when the def and use actions can happen to different server controls in the ASP.NET pages (ASPX file and code-behind class), then compute the def-use chains for them in order to derive suitable test cases. Table 1 shows the def and use actions that are defined for some ASP.NET server controls.

It should be noted that, a use action is set for the Data control and its defined properties, when it is used, as shown in the last row of Table 1. Following are some examples of the defs and uses of the ASP.NET server controls defined in Table 1.

Example 1: Defs and uses of a **LinkButton** control
The following definition element contains a ***def*** for a LinkButton control, named *SaveButton*:

```
<asp:LinkButton ID="SaveButton"
onclick="SaveButton_Click" Text="SaveChanges"
runat="server"></asp:LinkButton>
```

and the following header of the **Click** event handler of *SaveButton* control contains a ***use*** for that control:

```
public void SaveButton_Click(object
sender,System.EventArgs e)
```

Example 2: Defs and uses of a **TextBox** control and its **Text** property

The following definition element contains a ***def*** for a **TextBox** control, named *MyTextbox*:

Table 1. The def and use actions of some ASP.NET server controls

ASP.NET Data Item	Examples	Action	Statement	Location
Simple server control	TextBox, Button, Label, HyperLink, RadioButton, CheckBox	def	Definition element: <asp:ControlType ID= ControlName >	ASPX file
		use	header of an event handler for the control	Code-behind class
Simple server control property	Text	def	L.H.D. of an assignment statement	Code-behind class
		use	R.H.D of an assignment or Conditional statement	
List control	ListBox, DropDownList, CheckBoxList, RadioButtonList, BulletedList	def	Definition element: <asp:ListType ID= ListName>	ASPX file
			ListName.Items.Add();	Code-behind class
		use	header of an event handler for the list control	Code-behind class
List control properties	SelectedValue, SelectedItem, SelectedIndex	def	Statement that assign a value to the List property	Code-behind class
		use	Statement that reference the List property	
Data control	DataGrid, GridView	def	Definition element: <asp:datagrid ID= MyGrid >	ASPX file
		use	Header of an event handler for the data control	Code-behind class
Data control property	Columns	def	MyGrid.Columns.Add();	Code-behind class
	DataSource		L.H.D. of an assignment statement	
Data control and its properties	DataGrid and its properties Columns, DataSource	use	MyGrid.DataBind();	

```
<asp:textbox ID="MyTextbox" runat="server"  … >
</asp:textbox>
```

The following assignment statement contains a ***def*** for the property **Text** of *MyTextbox* control:

```
MyTextbox.Text = anyvalue;
```

and the following assignment statement contains a ***use*** for this property:

```
string str = MyTextbox.Text.ToString();
```

Example 3: Defs and uses of a **DropDownList** control

The following definition element contains a ***def*** for a DropDownList control, named *FavoriteLanguage*:

```
<asp:DropDownList ID="FavoriteLanguage"
runat="server">
```

```
<asp:ListItem Value=”C#”>C#</asp:ListItem>
<asp:ListItem Value=”Visual Basic”>Visual
        Basic</asp:ListItem>
<asp:ListItem Value=”CSS”>CSS</asp:ListItem>
</asp:DropDownList>
```

The following **Add()** method of *FavoriteLanguage* control, which adds an item to it, contains a ***def*** for that control:

```
FavoriteLanguage.Items.Add(item);
```

The following assignment statement, which refers to the currently selected item of *FavoriteLanguage* control, contains a ***use*** for the property SelectedValue of that control:

```
string language =
    FavoriteLanguage.SelectedValue.ToString();
```

Example 4: Defs and uses of a **DataGrid** control and its **Columns** and **DataSource** properties

The following definition element contains a ***def*** for a DataGrid control, named *MyDataGrid*:

```
<asp:datagrid ID="MyDataGrid" runat="server"
OnItemCommand = "MyDataGrid_Command" ...>
```

and the following header of the **Command** event handler of *MyDataGrid* control contains a ***use*** for that control:

```
protected void MyDataGrid_Command(object source,
    DataGridCommandEventArgs e)
```

The following **Add()** method of the **Columns** property of *MyDataGrid* control, which adds an item to it, contains a ***def*** for that property:

```
MyDataGrid.Columns.Add(value);
```

Also, the following assignment statement contains a ***def*** for the **DataSource** property of *MyDataGrid* control:

```
MyDataGrid.DataSource = ds;
```

where ds is a DataSet object. The following **DataBind()** method of *MyDataGrid* control, which binds the control and all its child controls to the specified data source, contains a ***use*** for that control, and its Columns and DataSource properties:

```
MyDataGrid.DataBind();
```

5.1.3 Defs and uses of implicit session/state objects

The implicit session/state object, such as Query-String parameters, ViewState property, and Session property of ASP.NET pages, is like a container that can store name-value pairs. Through the session/state object, the values of the name-value pairs can be used or changed by different ASP.NET pages. This enables data to be accessed across ASP.NET pages within a user session and, hence, introduces data interactions between ASP.NET pages.

5.1.3.1 QueryString parameters

There are few ways to create links between pages or redirect the user to another page, including: using the HTML <a> element, using the HyperLink Web server control, or programmatically through code, using Response.Redirect() and Server.Transfer() methods [12]. Quite often when the user is to be sent to a different page, using one of these ways, some additional information has to be sent along. This can be done by passing it in the query string. The query string is the part of the address that comes after the page name separated by a question mark (?). It consists of name-value pairs, each separated from another by an ampersand (&). Consider the following URL:

Target.aspx?**CategoryId=10&From=Home**

The entire bold part (after the question mark) is considered the query string. In this case, the query string consists of two pairs: *CategoryId* with a value of *10* and *From* with a value of the word *Home*. The page, Target.aspx in this example, is able to read these values using Request.QueryString, as in the following example:

```
Label1.Text = Request.QueryString.ToString();
```

Also, the page is able to get the value of any these values using Request.Params with the name of the required value as an index, as in the following example:

```
Label1.Text = Request.Params["CategoryId"].ToString();
```

where Params is of type NameValueCollection. Request.Params can return name-value pairs from: Query-string parameters, Form fields, Cookies, and Server variables, in that order [13].

In the proposed data flow testing approach, the name part of the name-value pair is treated as a global variable and the value part as the value assigned to it. So, a ***def*** action is set for this variable at the statement, which specifies the query string, such as <a> element, the <asp:Hyperlink> control, and Response.Redirect() and Server.Transfer() methods. This global variable can be used by the methods in the code-behind file of the target page. For example, at the following statement, a ***def*** action is set for the name part, "Test", of the query string Test=SomeValue:

```
Response.Redirect("Target.aspx?Test=SomeValue");
```

The value part of the query-string Test=SomeValue can be retrieved in the target page Target.aspx as follows:

```
TestValue = Request.Params["Test"];
```

At this statement, a ***use*** action is set for the name part, "Test", of the query string.

5.1.3.2 ViewState property

The ViewState collection is a property on the Page class. It is a bag that enables storing data that can be retrieved again after a postback. Values in ViewState can be identified using a unique key. The values stored in this collection it gets sent to the browser when the page loads and it is sent back to the server when the page is posted back again. The ViewState is stored within the page. The following example illustrates the def and use actions that can be performed on the ViewState property.

Example 5:
Assume that the following <a> element is defined in an ASPX file:

```
<a href="ToDoList.aspx?query=0">Show All Items</a>
```

In this element, there is query string: "query=0". So, the name "query" is considered as a global variable, and a ***def*** action is set for it.

Also, assume that the following code is defined in the related code-behind class:

```
1   int x;
2   if(!IsPostBack)
3   {
4   string queryStr = Request.Params["query"];
5   x = Int32.Parse(queryStr);
6   ViewState["query"] = x;
7   }
8   else
9   x = (int)ViewState["query"];
```

In this example, the global variable query, defined above in the <a> element, is referenced at line 4. So, a ***use*** action is set for it. Also, at line 6, the variable query is referenced and ViewState is assigned a value. So, a ***use*** action is set for query, and a ***def*** action is set for ViewState. At line 9, both the variable query and ViewState are referenced. So, a ***use*** action is set for each of them.

5.1.3.3 Session Property

Every ASP.NET page has a property Session, which provides information about the current session. It stores session variables (key/value pairs) that can be accessed by any page during the same session. The value in a key/value pair is retrieved from the Session property by indexing the Session property with the key name [14]. So, the Session property is considered as a global variable. The following example illustrates the def and use actions that can be performed on the Session property.

Example 6:
Assume that a *RadioButtonList* control, named *languageList*, is defined in an ASPX file of a page as follows:

```
<asp:RadioButtonList ID="languageList" runat="server">
<asp:ListItem Value="0-13-605322-X">Visual C#
</asp:ListItem>
        . . .
<asp:ListItem Value="0-13-605306-8">Java
</asp:ListItem>
</asp:RadioButtonList>
```

At this element, there is a ***def*** for *languageList* control.

Also, assume that the following code is defined in the related code-behind class:

```
1   if ( languageList.SelectedItem != null )
2    // add name-value pair to Session
3    Session.Add( languageList.SelectedItem.Text,
          languageList.SelectedItem.Value );
```

At lines 1 and 3, there are ***uses*** for the property *SelectedItem* of *languageList* control, and at line 3 there is a ***def*** for the *Session* property, as a key/value pair is added to it, and there is a ***use*** for *languageList* control.

Table 2. The def and use actions of the objects of some SQL Server classes

SQL Server Class	Description	Action	Affected Class Object	Example
SqlConnection	Represents a unique session to a SQL Server database	def	object created using the class constructor	SqlConnection con = new SqlConnection (conStr); - def of con
		use	object calls one of the class methods, such as Open, Close or Dispose	con.Open(); - use of con
			Object referenced in any statement	See next example
SqlCommand	Represents a Transact-SQL statement or stored procedure to execute against a SQL Server database	def	object created using the class constructor	SqlCommand cmd = new SqlCommand (queryString, con); - def of cmd - use of con
		use	object calls one of the class methods to execute commands at a SQL Server database	cmd. ExecuteNonQuery(); - use of cmd
SqlDataAdapter	Represents a set of data commands and a database connection that are used to fill a DataSet object and update a SQL Server database	def	object created using the class constructor	SqlDataAdapter adapter = new SqlDataAdapter(); - def of adapter

SqlDataAdapter Class properties	SqlDataAdapter Class has properties, such as SelectCommand, DeleteCommand, and InsertCommand	def	property assigned a SqlCommand object to set a Transact-SQL statement to manipulate records in the data source	adapter.SelectCommand = new SqlCommand (queryString, con); - def of adapter.SelectCommand
SqlDataAdapter Class and its properties		use	object and its defined properties, when the object calls one of the class methods to fill a DataSet object or update a database	adapter.Fill(dataset); - use of adapter - use of adapter.SelectCommand

Also, assume that a *ListBox* control, named *booksListBox*, is defined in the ASPX file of another page in the same website as follows:

<asp:ListBox ID="booksListBox" runat="server" CssClass="listBoxStyle">

</asp:ListBox>

At this element, there is a ***def*** for *booksListBox* control.

Then, assume that the following code is defined in the related code-behind class:

```
1   // determine whether Session contains any
    // information
2   if ( Session.Count != 0 )
3   {
4      // display Session's name-value pairs
5      foreach ( string keyName in Session.Keys )
6         booksListBox.Items.Add( keyName +
                " ISBN#: " + Session[keyName] );
7   } // end if
```

At lines 2, 5 and 6, there are ***uses*** for Session property, as it is referenced in Session.Count, Session.Keys and Session[keyName], respectively. At line 6 there is a ***def*** for booksListBox control, as an item is added to it.

5.1.4 Objects of Built-In Classes

In Web applications, objects of built-in classes, such as SQL Server classes and custom button controls, can be created and used in the code-behind class. The following subsections explain how def and use actions are set for the objects of some of these classes.

5.1.4.1 SQL Server classes and ADO.NET library classes

Many Web applications allow users to provide/get certain information through Web forms, which include ASP.NET data controls (such as DataGrid or GridView), and store/retrieve this information in/from a SQL Server database located on the Web server. This requires setting up data binding between the data control and the database and allowing interactions between them. This can be done programmatically by using objects of certain SQL Server classes and ADO.NET library classes. Tables 2 and 3 show, with examples, the def and use actions that are set for the objects of the SQL Server classes: SqlConnection, SqlCommand, and SqlDataAdapter, and ADO.NET classes: DataSet, and DataTable, respectively. It should be noted that, a *use* action is set for the SqlDataAdapter object and its defined properties, when it calls one of the class methods to fill a DataSet object or update a database, as shown in Table 2.

5.1.4.2 Custom button controls

The ButtonColumn class is used to display a command button for each item in a column in a DataGrid control [15]. This allows the creation of a column of **custom button controls**, such as **Add** or **Remove** buttons.

A ***def*** action is set for a ButtonColumn object, when it is created using its constructor, and also a ***def*** action is set for any one of its properties, such as CommandName, DataTextField, HeaderText, and Text, when a value is assigned to that property. A ***use*** action is set for the ButtonColumn object and its defined properties, when it is added to a DataGrid control using the Add() method.

To illustrate the settings of def and use actions for ButtonColumn class objects, consider the following example, which uses the ButtonColumn class in a DataGrid control to create an ***Add*** button.

```
1.  private void Page_Init(Object sender, EventArgs e)
2.  {
3.     // Create dynamic column to add to Columns
       // collection.
4.     ButtonColumn AddColumn =
       new ButtonColumn();
5.     AddColumn.HeaderText="Add Item";
6.     AddColumn.Text="Add";
7.     AddColumn.CommandName="Add";
```

Table 3. The def and use actions of the objects of some ADO.NET classes

ADO.NET Class	Description	Action	Affected Class Object	Example
DataColumn	Represents the schema of a column in a DataTable	def	object created using the class constructor	DataColumn idColumn = new DataColumn(); - def of idColumn
		use	object added to a DataTable, using the Add() method	See next example
DataTable	Represents one table of in-memory data, and used by objects of the DataSet and the DataView classes	def	object created using the class constructor	DataTable table = new DataTable ("ParentTable"); - def of table
			object that a column or row is added to, using the Add() method	table.Columns.Add (idColumn); - def of table - use of idColumn
		use	object added to a DataSet, using the Add() method	dataSet.Tables.Add (table); - use for table
			object that one of its properties is referenced in any statement	If(tab1e.Rows.Count > 0) - use of table
DataRow	Represents a row of data in a DataTable	def	object created using the NewRow() method of DataTable	DataRow row = table.NewRow(); - def of row
			object assigned a reference to a row of a DataTable object	DataRow row = tab1e.Rows[0]; - def of row - use for table
		use	object added to a DataTable, using the Add() method	table.Rows.Add (row); - def of table - use of row
			object referenced in any statement	IdTextbox.Text = row["id"].ToString(); - use of row
DataSet	Represents an in-memory cache of data retrieved from a data source and consists of a collection of DataTable objects	def	object created using the class constructor	DataSet dataSet = new DataSet(); - def of dataSet
			object that a DataTable object is added to, using the Add() method	dataSet.Tables.Add (table); - def of dataSet
			object filled with data from a data source using a DataAdapter object	adapter.Fill(dataset); - def of dataSet
		use	object referenced in any statement	MyDataGrid. DataSource = dataset; - use of dataSet

8. AddColumn.ButtonType =
   ButtonColumnType.PushButton;
9. // Add column to Columns collection.
10. ItemsGrid.Columns.Add(AddColumn);
11. }

At line 4, there is a ***def*** for the *ButtonColumn* object, *AddColumn*. At lines 5-8, there are ***defs*** for *AddColumn* properties: HeaderText, Text, CommandName, and ButtonType, respectively. At line 10, *AddColumn* is referenced. So, a ***use*** action is set for it and for each one of its defined properties. In addition, there is a ***def*** for the DataGrid object, ItemsGrid, at line 10.

5.2 Data Flow Testing Levels of Web Applications

Since a Web application consists of one or more Web pages and each Web page consists of an ASPX file and a code-behind class, in the proposed Web applications data flow testing approach, four testing levels are considered, ***Function, Interprocdural***, ***Page***, and ***Inter-Page*** levels. These testing levels are described below.

5.2.1.1 Function Level Testing

Function level testing is used to test the variables that have def-use chains limited to a single function. To test a function, the def-use chains for this function are obtained from its CFG, then test cases are generated to cover them.

5.2.1.2 Interprocdural Level Testing

Interprocdural level testing is used to test the variables whose def-use chains involve more than one function that interact through function calls within a page. The Interprocdural level testing is required if a variable has a definition in one function and uses of this definition in other functions within a page. To test interacting functions through function calls within a page, the def-use chains for them are computed from their ICFG, then test cases are generated to cover them.

5.2.1.3 Page Level Testing

Page level testing is used to test the instance variables, global variables, and session/state variables that have def-use chains limited to a single page (ASPX file and code-behind class). To test a page, the def-use chains for its variables are computed from its PCFG, then test cases are generated to cover them.

5.2.1.4 Inter-Page Level Testing

Inter-Page level testing is used to test the variables whose def-use chains cross Web pages in the Web application either through function calls or session/state variables. In the case of passing data through function calls, def-use chains for the inter-page level can be derived from the ICFG of involved functions in the Web application pages. In the case of passing data through session/state variables, such as data transmissions between Web pages via Response.Redirect() or Server.Transfer() methods, def-use chains can be computed from the CCFG of related Web pages. Then test cases are generated to cover the obtained def-use chains.

6. CASE STUDY

To illustrate the proposed Web applications data flow testing approach, it is applied to an example ASP.NET Web page, named EditItem.aspx, which is a part of a Web application, named 'To Do List', for a simple personal agenda. This application was adopted from [16]. The ASPX file and the related C# code-behind class of the EditItem page are listed in Fig. 1.

// The ASPX file of the EditItem Page

```
0.  <%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="EditItem.aspx.cs" Inherits="EditItem" %>
1.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    transitional.dtd">
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head runat="server">
4.    <title>
5.      <%=_title2%>
6.    </title>
7.    <style type="text/css">
        …
11.   </style>
12. </head>
13. <body>
14.   <h1><%=_title2%></h1>
15.   <form id="EditItemForm" method="post"
                                        runat="server">
16.     Description<br/>
17.     <asp:textbox ID="DescriptionTextbox" runat="server"
         Font-Names="Verdana" Font-Size="8pt"
         Width="100%"></asp:textbox>
18.     <br />
19.     Priority<br />
20.     <asp:DropDownList ID="PriorityList" Font-
         Names="Verdana" Font-Size="8pt"
         runat="server"></asp:DropDownList><br />
21.     <asp:Label ID="ErrorLabel" runat="server" Text=""
         Visible="false" ForeColor="Red"></asp:Label>
         <br />
22.     <asp:LinkButton ID="SaveButton"
         onclick="SaveButton_Click" Text="SaveChanges"
         runat="server"></asp:LinkButton>
        …
25.     <asp:LinkButton ID="LinkButton1" runat="server"
         onclick="LinkButton1_Click">return
26.     </asp:LinkButton>
27.   </form>
28. </body>
29. </html>
```

// The C# code-behind class of the EditItem Page

```
30. using System;
31. using System.Collections;
    …
45. public partial class EditItem : System.Web.UI.Page
46. {
47.   protected string _title2;
48.   public string connStr = ConfigurationManager.
        ConnectionStrings["ConnectionString"].
        ConnectionString;
49.   protected static string[] _priorites = { "Low", "Medium",
        "High" };
50.   public string idStrED;
51.   protected void Page_Load(object sender, EventArgs e)
52.   {
53.     SqlConnection con = new SqlConnection(connStr);
54.     idStrED = Request.Params["id"];
55.     _title2 = (idStrED == null ? "New" : "Edit") +
                  "ToDoListItem";
56.     if (!IsPostBack)
57.     {
58.       foreach (string s in _priorites)
59.       {
60.         PriorityList.Items.Add(s);
61.         if (idStrED != null)
```

Fig 1: The ASPX file and the C# code-behind class of the EditItem Page

```
62.             {
62.               con.Open();
63.               string queryStr = "select * from items where
                        id=" +idStrED;
64.               SqlDataAdapter ad =
                        new SqlDataAdapter(queryStr, con);
65.               DataSet ds = new DataSet();
66.               ad.Fill(ds);
67.               DataTable tb1 = ds.Tables[0];
68.               if (tb1.Rows.Count > 0)
69.               {
70.                 DataRow row = tb1.Rows[0];
71.                 PriorityList.SelectedValue =
                                row["priority"].ToString();
72.                 DescriptionTextbox.Text =
                                row["description"].ToString();
73.               }
74.               con.Close();
75.             }
76.           }
77.       }
78.     }
79.     public void SaveButton_Click(object
                    sender,System.EventArgs e)
80.     {
81.       OnSubmit(idStrED);
82.     }
83.     private void OnSubmit(string f)
84.     {
85.       SqlConnection con = new SqlConnection(connStr);
86.       con.Open();
87.       string idStrF=Request.Params["id"];
88.       string desc = DescriptionTextbox.Text.ToString();
89.       int priorityN=PriorityList.SelectedIndex+1;
90.       string priority = PriorityList.SelectedValue.ToString();
91.       SqlCommand cmd = new SqlCommand();
92.       if ((f == null) || (idStrF == null))
93.       {
94.         cmd = new SqlCommand("insert into items
                (description,priority) values ('" + desc + "'," +
                priority + ")", con);
95.         cmd.ExecuteNonQuery();
96.         con.Close();
97.       }
98.       else
99.       {
100.        cmd = new SqlCommand("update items set
                description='" + desc + "',priority='" + priority +
                "' where id=" + f, con);
101.        cmd.ExecuteNonQuery();
102.        con.Close();
103.      }
104.      con.Close();
105.    }
106.    protected void LinkButton1_Click(object sender,
                    EventArgs e)
107.    {
108.      Response.Redirect("ToDoList.aspx");
109.    }
110. }
```

Fig 1: The ASPX file and the C# code-behind class of the EditItem Page (Continued)

The first step in the proposed Web applications data flow testing approach is to perform data flow analysis of the example Web page. This step includes two tasks: constructing the Web application data flow model for the example Web page, as described in Sec. 4, then collecting information about the defs and uses of the page variables, as described in Sec. 5. Fig. 2 shows the CFGs of the components of example Web page, annotated with defs and uses of the page variables.

From the collected data flow information, the def-use chains of Web application variables are computed, then the list of du-paths of the example Web page are constructed as described in [11]. Table 4 shows the list of the constructed du-paths. In this list, each du-path is represented by: a def-node (a node containing a def of a variable); a use-node (a node containing a use of that variable); and the set of nodes that must not be included in that path (nodes containing other defs of that variable). These nodes are called killing nodes [11]. The value (-1) is used in the killing node column to indicate that the du-path has no killing nodes.

After determining the list of du-paths for the program being tested, the tester runs the program with test data. During each test run, the traversed path is recorded. At the end of a test run, the traversed path is checked to see whether it covers any of the constructed du-paths, and a report is produced to the tester showing the du-paths that have not been covered yet. A path is said to cover a du-path if it has a subpath that starts at the def-node and ends at the use node of the du-path and does not pass through its killing nodes [11]. The tester then re-runs the program with different test data until all the remaining du-paths are covered, unless there are some infeasible paths that cannot be covered by any test data.

Fig. 3 shows the report produced by the system, which have been developed to implement the proposed approach, after a test run of the example application. It shows the traversed path, the covered du-paths, and the du-paths that are not covered yet. More test runs are performed until the remaining du-paths are covered, if possible.

7. CONCLUSION

In this paper, an approach to data flow testing of ASP.NET Web applications has been proposed. This approach consists of three steps: The first step is the construction of a Web application data flow model to support data flow analysis of ASP.NET Web applications. This model consists of four types of flow graphs: control flow graph (CFG), interprocedural control flow graph (ICFG), page control flow graph (PCFG), and composite control flow graph (CCFG). The second step is the data flow analysis of the Web application to be tested, to collect information about the defs and uses of its variables. Five types of variables/data objects, which can be found in ASP.NET Web applications, have been considered: traditional program variables and arrays, instance variables of the code-behind class, simple and complex ASP.NET server controls and their properties, implicit session/state objects, and objects of built-in classes, such as SQL server classes. The third step is the computation of the def-use chains of the Web application variables, using the collected data flow information, with the constructed Web application data flow model, and the data flow testing technique described in [11]. In the proposed approach, testing is performed in four levels, *Function*, *Interprocdural*, *Page*, and *Inter-Page* levels. In each level, the def-use chains of the variables of interest are computed, then test data can be generated to cover these def-use chains, in order to fulfill one of the data flow coverage criteria, namely the all-uses criterion. Finally, the application of the proposed approach has been illustrated through a case study.

Currently, experiments are being conducted to evaluate the effectiveness of the proposed approach in the structural testing of Web applications, and to evaluate its ability to expose different types of errors that may occur in the code-behind and ASPX files of Web applications.

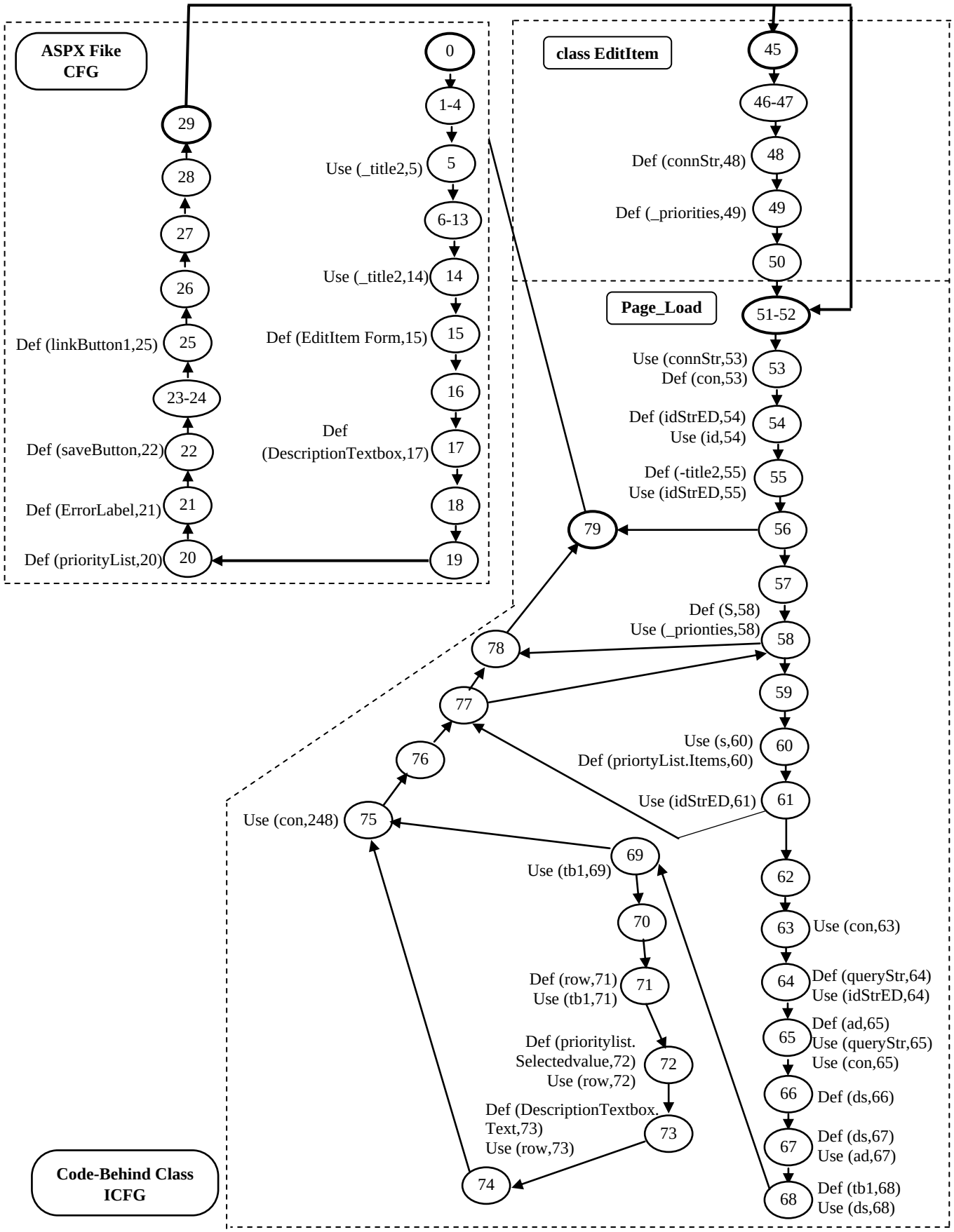

Fig 2. The CFGs of the components of example Web page

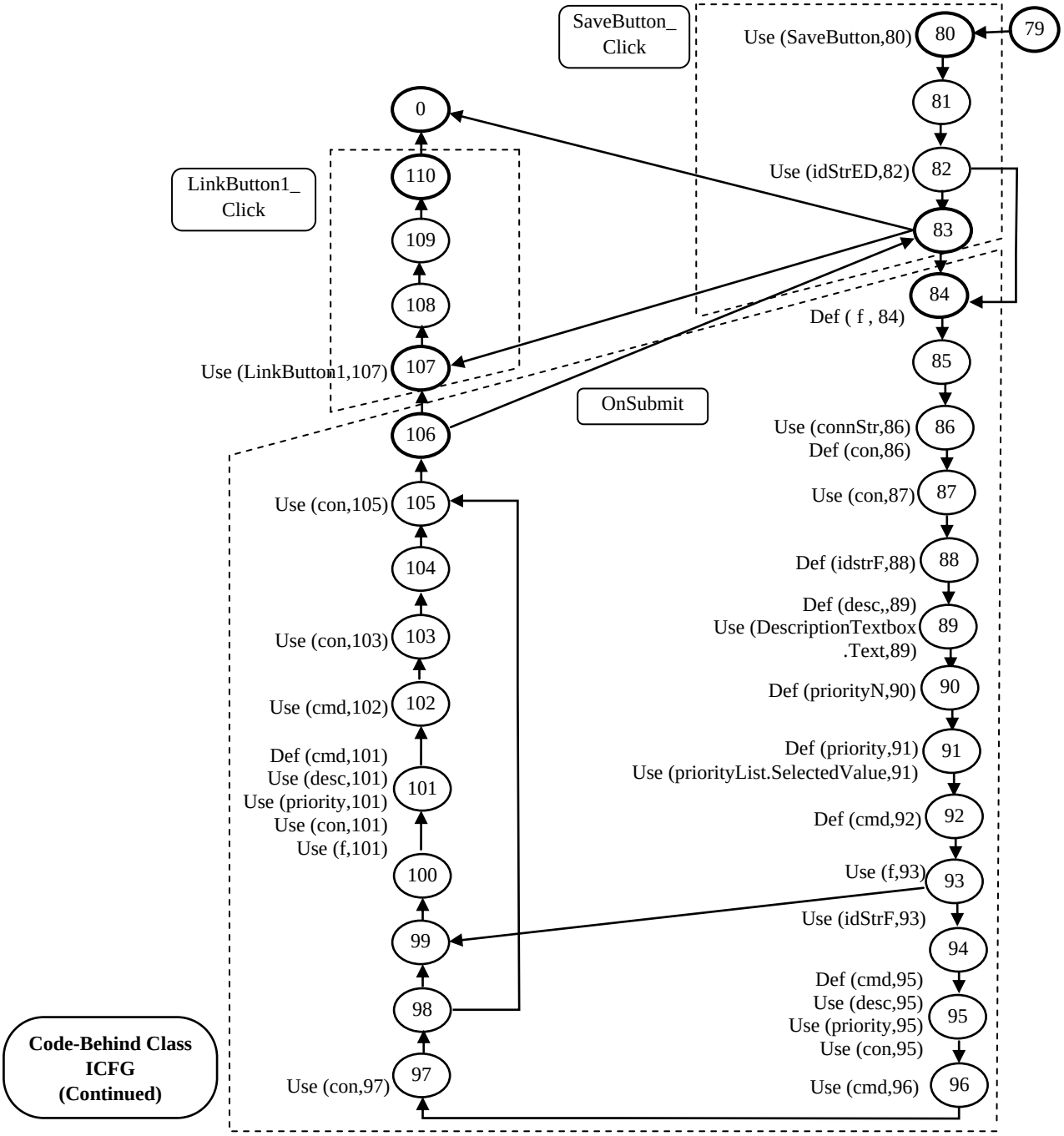

Fig 2: The CFGs of the components of example Web page (Continued)

******** RUN (1) ********
PATH NUMBER: 1
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 77 58 59 60 61 77 58 59 60 61 77 78 79 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 52 53 54 55 56 79
THE NEWLY DU-PATHS COVERED BY THIS PATH:
<(_title2,55),5>, <(connStr,48),53>, <(idStrED,54),55>, <(_priorites,49),58>, <(s,58),60>, <(idStrED,54),61>, <(_title2,55),14>, <(SaveButton,22),80>, <(idStrED,54),82>, <(connStr,48),86>, <(con,86),87>, <(f,84),93>, <(idStrF,88),93>, <(con,86),95>, <(desc,89),95>, <(priority,91),95>, <(cmd,95),96>, <(con,86),97>, <(con,86),105>

THE DU-PATHS NOT COVERED YET
<(con,53),63>, <(idStrED,54),64>, <(con,53),65>, <(queryStr,64),65>, <(ad,65),67>, <(ds,67),68>, <(tb1,68),69>, <(tb1,68),71>, <(row,71),72>, <(row,71),73>, <(con,53),75>, <(f,84),101>, <(con,86),101>, <(desc,89),101>, <(priority,91),101>, <(cmd,101),102>, <(con,86),103>, <(LinkButton1,25),107>

Fig 3: The Report of the fulfillment of all-uses criterion for the example Web page after a test run

Table 4. The list of du-paths and killing nodes of the example Web page

No.	Variable	Def node	Use node	Killing Nodes	No.	Variable	Def node	Use node	Killing Nodes
0	_title2	55	5	-1	20	idStrED	54	82	-1
1	_title2	55	14	-1	21	connStr	48	86	-1
2	connStr	48	53	-1	22	con	86	87	-1
3	idStrED	54	55	-1	23	f	84	93	-1
4	_priorites	49	58	-1	24	idStrF	88	93	-1
5	s	58	60	-1	25	con	86	95	-1
6	idStrED	54	61	-1	26	desc	89	95	-1
7	con	53	63	-1	27	priority	91	95	-1
8	idStrED	54	64	-1	28	cmd	95	96	92, 101
9	con	53	65	-1	29	con	86	97	-1
10	queryStr	64	65	-1	30	f	84	101	-1
11	ad	65	67	-1	31	con	86	101	-1
13	ds	67	68	66	32	desc	89	101	-1
14	tb1	68	69	-1	33	priority	91	101	-1
15	tb1	68	71	-1	34	cmd	101	102	92, 95
16	row	71	72	-1	35	con	86	103	-1
17	row	71	73	-1	36	con	86	105	-1
18	con	53	75	-1	37	LinkButton1	25	107	-1
19	SaveButton	22	80	-1					

8. REFERENCES

[1] Liu, C., Kung, D. C., Hsia, P., and Hsu, C. 2000 Object-based data flow testing of Web applications. In Proceedings of the First Asia-Pacific Conference on Quality Software.

[2] Liu, C. H., Kung, D. C., Hsia P. 2001. Object-based data flow testing of Web applications. Int. J. Soft. Eng. and Know. Eng. 11 (April 2001), 157-179

[3] Lee, S. C. and Offutt, J. 2001 Generating test cases for XML-based Web component interactions using mutation analysis. In Proceedings of the 12th International Symposium on Software Reliability Engineering.

[4] Ricca, F., Tonella, P. 2001 Analysis and testing of Web applications. In Proceedings of the International Conference on Software Engineering.

[5] Ricca, F., Tonella, P. 2004 A 2-layer model for the white-box testing of Web applications. In Proceedings of the Sixth IEEE Workshop on Web Site Evolution.

[6] Mansour, N., and Houri, M. 2006. Testing Web applications. Information and Software Technology. 48 (2006), 31–42.

[7] Qi, Y., Kung, D., and Wong, E. 2006. An agent-based data-flow testing approach for Web applications. Information and Software Technology. 48 (2006), 1159–1171.

[8] Liu, C.-H. 2006. Data flow analysis and testing of JSP-based Web applications. Information and Software Technology. 48 (2006), 1137–1147.

[9] Frankl, P. G. and Weiss, S. 1993. An Experimental Comparison Of The Effectiveness Of Branch Testing And Data Flow Testing. IEEE Trans. on Soft. Eng. 19 (August 1993), 774-787.

[10] Rapps, S. and Weyuker, E. J. 1985. Selecting software test data using data flow information. IEEE Trans. on Soft. Eng. 11 (1985), 367-375.

[11] Girgis, M. R. 1993. Using Symbolic Execution and Data Flow Criteria to Aid Test Data Selection. Software Testing, Verification and Reliability. 3 (1993), 101-112.

[12] Spaanjaars, I. 2010. Beginning ASP.NET 4 in C# and VB. Wiley Publishing, Inc.

[13] msdn.microsoft.com, 2016. HttpRequest.Params Property. https://msdn.microsoft.com/en-us/library/system.Web.httprequest.params(v=vs.110).aspx

[14] Deitel, P. J. and Deitel, H. M. 2009. Visual C# 2008 How to Program. 3rd Edition, Pearson Education Inc.

[15] msdn.microsoft.com, 2016. DataGridColumnCollection Class. https://msdn.microsoft.com/en-us/library/system.Web.ui.webcontrols.datagridcolumncollection(v=vs.110).aspx.

[16] Lyon-Smith, J. 2002. ASP.NET to Do List Application. http://www.codeproject.com.