Chapter 7: System Testing

7.1 Testing Approach

The goals of software testing are to establish the functionality of a system as well as to detect the presence of errors or inaccuracies in the system. There are various approaches to testing; each type of testing validates different aspects of the software. The selection of the type of testing depends on the nature of the software system. The initial stages of testing should be the responsibility of the programmers, testing the codes they have developed and coded. When this stage is completed, the project is handed over to an independent team to perform integration testing.

During the process of testing, the software requirements must be made available, and tests are then designed based upon these requirements. However, testing should not rely totally on the software specifications, as the development of software is never a static process. There are various reasons for this phenomenon to happen:

- Requirements may be modified without updating the specification.
- As the development project progresses, new and better ways may have been found since the initiation of the software process, that can solve the problem, usually is a better way.
- Time constraints in the release of the software could result in the use of 'shortcuts' that bypasses some routines, which were deemed superfluous or unnecessary.
- Certain unforeseen constraints in the hardware or software may prevent the exact implementation of the software specification. For example, certain functionality may perform in a manner that may be too unresponsive owing to the hardware available. Certain functionality may not be possible to be programmed in the selected programming language.
Differences in implementation environment may demand alterations in the software functionality. A system may perform perfectly well in the development hardware and software environment. However, when the system is implemented in the customer’s site, severe errors may arise owing to certain factors that are not present in the development environment. This could arise from myriad causes:

- The customer site uses a different processor, that is, there is a hardware conflict.
- At the customer site, they may have installed other types of software, which conflict with the system being implemented.
- The usage load could vary. Other systems running at the customer site may already have a certain degree of usage pattern or usage load that cause performance problems with the system that is implemented.
- Switch in Build Configuration. During testing, the systems may have been built using the Debug Configuration. Once the testing has been completed, it is changed to Release Configuration. Certain routines perform well in the Debug Configuration, but the performance deteriorates, or even fails when switched to the Release Configuration. This switch could result in a deviation from the specification in order to get the system functioning again.

All these factors have to be considered in the planning of the testing stages, failing which the testing would be incomplete.

7.2 Types of Testing

The types of testing that are to be executed depends on the stage in the development and testing. The process EWSS testing follows a pattern that can be shown in the Figure 7.1.

The first stage in the testing process involve unit testing. There could be multiple units to test. Hence the multiple boxes in the diagram. After this, there is the module
testing stage, which can involve multiple modules to be tested. If any of these modules are defective, it is necessary to return to unit testing. However, this return path is optional, and is performed only if the module testing failed. The dotted arrow indicates this optional path. This is followed by Interface testing, to test the communication between the various modules. When all these are done, the integration phase begins, combining all the individual modules together. Finally the whole system is tested.

![Diagram](image)

*Figure 7.1 Testing Process*

### 7.2.1 Unit Testing

This stage aims at testing the smallest components of the code. The initial units in this project are the individual Web Services, each of which may consist of several units. The tests are done by using the Visual Studio.NET debugger. A typical screen is shown in Figure 7.2.
This type of screen has been very useful in the unit testing of the individual web services. There are several approaches to unit testing, each of which has their own strengths and weaknesses. These are White box testing and black box testing. This is defect testing.

**Figure 7.2 Black Box Testing of a Web Service**

7.2.1.1 Black Box testing

This is a functional testing procedure to establish whether the unit meets its specifications. For example, in the screen in Figure 7.2, the test allows the entry of two inputs, the Class and the Form, which are the parameters to the getMarkSheet web service. The output is then studied, which may look like Figure 7.3. This is part of the XML output from black box testing of a web service. The result is a successful output, indicating a successful test. In Black Box testing, we are only concerned about what inputs that are put in, and what outputs are generated as a result of the test. Black box
testing is only concerned about functionality. If the output is incorrect, or there is no data output, the test has failed.

Figure 7.3 Successful Output of Black Box Testing

7.2.1.2 White Box Testing

If the result of a black box testing indicates a failure, there are other approaches to rectify the problem. Here the structural testing approach is needed, where the code is analyzed. Knowledge of the structure of the component is vital in white box testing.

In Figure 7.4, the structure of a unit in the Web Service is examined:
public DataSet getMarkSheet(string ClassCode, int FormNo)
{
    SqlCommand sqlCmd = new SqlCommand();
    DataSet ds = new DataSet();
    sqlCmd.CommandType = CommandType.StoredProcedure;

    switch (FormNo)
    {
    case 1:
        sqlCmd.CommandText = "sp_getT1ClassMarkSheet";
        break;
    case 2:
        sqlCmd.CommandText = "sp_getT2ClassMarkSheet";
        break;
    case 3:
        sqlCmd.CommandText = "sp_getT3ClassMarkSheet";
        break;
    case 4:
        sqlCmd.CommandText = "sp_getT4ClassMarkSheet";
        break;
    case 5:
        sqlCmd.CommandText = "sp_getT5ClassMarkSheet";
        break;
    default:
        // to handle later
        break;
    }

    try
    {
        SqlParameter myParam = new SqlParameter("@ClassCode", SqlDbType.NVarChar)
        myParam.Value = ClassCode;
        sqlCmd.Parameters.Add(myParam);
        sqlCmd.Connection = sqlConnection1;
        SqlDataAdapter myAdapter = new SqlDataAdapter(sqlCmd);
        myAdapter.Fill(ds, "ClassMarkSheet");
    }
    catch (SqlException ex)
    {
        Console.WriteLine(ex.ToString());
        throw;
    }
}

7.4 White Box Testing: Examining the Code
7.2.1.3 Code Flow

It is very often that it is necessary to examine the flow, or path of a code segment. This could lead to a deeper and deeper examination of the flow of the logic of the code. Path testing is also a part of examining code flow, which traces the flow of the program from one point to another. Here, we examine the flow of the code through looking at the stored procedure, in order to discover any defects in the coding:

Figure 7.5 Examining Code Flow

7.2.2 Module Testing

When each of the units have been tested, it is necessary to ensure that the modules work together. Such is the case of the web service as one unit. Then there is the Client Module to obtain the results from the web service. The Client module consists of several units that are combined together. In the diagram below, it consists of several Windows Forms controls and two procedures. Each of these units must have been tested earlier, and combined together to form the Client Module. Each unit does not work alone, as each unit depends on another unit. For example, here, Procedure 1, that is, to enter marks, need to obtain the ‘Form’ data from the radio button, and the ‘Class’ data from the ComboBox to form the two parameters to be passed to a Web Service, which is another external unit
or module. The testing must ensure that these data are passed to the correct procedures and activate a particular functionality. Figure 7.6 shows the modules being examined in the Marks Entry Screen.

![Form1](image)

**Figure 7.6 Units in a Client Module**

7.2.3 Interface Testing

Interface testing in this project is very important. Data is passed between Client and Web Server using XML SOAP messages. These data must conform to the same standards and must reach their destination in the form that is expected. In Interface testing, we need to know whether these messages actually are in the format that is expected by both ends. Figure 7.7 shows the interface between a Web Service module and the Client module that calls it.
7.2.4 Integration Testing

In integration testing, all the modules are combined together to test if they all work together as a system. This testing will expose any inconsistencies in each module, as well as inconsistencies in the integration itself.

![Diagram of Web Service and Client Module](image)

*Figure 7.7 Interface between Modules: Web Service and Client Module*

7.2.5 System Testing

The final step in the test is system testing, where a series of tests are conducted in order to exercise the whole system. The whole system includes both the interaction between the modules, as well as the interaction with the hardware. In this project, there is also the network factor, and the interaction between different computers in the network.

If the system works on one machine, can it work on other machines with different sets of hardware and software? Here it was necessary to test placing the Web Services in a Web Server on one machine, and testing how clients from other machines access the web services.
There is also the need to test for system performance, stress test, security access rights, usability, data integrity, error handling and recovery.

7.3 Acceptance of User Testing

A software system is a failure if there are no users. It was necessary to test the whole system in the school environment, where students and teachers will be accessing the system. It is necessary to convince the administration of the benefits of the system, compared to the current working practices. Some of this testing require inter-personal skills that go far beyond just software skills.¹

7.4 Performance Testing

Performance testing involves obtaining data concerning how well the system executes the functions they were designed for. Part of the performance involves a comparison of its performance with past projects of a similar functionality.

Figure 7.8 shows the evolution in the performance of examinations results processing software in Sekolah Menengah Chan Wa. There is an increase in the time of processing of the results of a class of 40 students in the transition from using a file processing, C-Language DOS program to the system using Web Services. This disparity in performance is almost not noticeable to the general user, and is really insignificant. This chart also demonstrates that even though the system uses Web Services, which involve so much more levels of software, especially when running over the Intranet system, gives a performance that is comparable to a very simple C-Language program. This chart does not represent a true performance of the software evolution, as the hardware platforms have changed dramatically over the years. However, this chart shows an overall performance as a combination of real existing factors at the time the types of processing were used.

¹ Section 8.2 documents the user feedback and user acceptance of EWSS. It has been accepted for implementation in CHANWA, with some request for implementation in other schools.
A number of tests were conducted to gauge the performance of the system in its use of web services. The aim of the tests is to determine and justify the type of interfaces used in each of the modules in EWSS. One of these test involve a run through the processing of examination records for all of Form Five. The performance test screens are shown in Figure 7.9 and 7.10 below. The results of the tests are shown in Figure 7.11. The test involves cycling through 395 rows in 11 Classes of Form Five Results.

<table>
<thead>
<tr>
<th>Year</th>
<th>Processing Time in seconds</th>
<th>Language</th>
<th>Hardware</th>
<th>Test Environment</th>
<th>Software</th>
<th>Time to process Results Of a Class of 40 students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>1200 BASE II</td>
<td>IBM PC Compatible 4.77 MHz</td>
<td>Interpretive</td>
<td>30 minutes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>120 Turbo Pascal</td>
<td>IBM PC Compatible 20 MHz</td>
<td>Compiled</td>
<td>2 minutes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1992</td>
<td>260 C</td>
<td>IBM PC Compatible 75 MHz</td>
<td>Compiled</td>
<td>2.6 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>10</td>
<td>IBM PC Compatible 90 MHz</td>
<td>Compiled</td>
<td>1.6 seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>140 C#</td>
<td>IBM PC Compatible 300 MHz</td>
<td>Compiled</td>
<td>1.4 seconds</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7.8 The evolution in the performance of examinations results processing software**

All the tests are conducted using the same algorithm as in Table 7.1

Figure 7.9 shows the performance test screen for ASP.NET Web Pages. Figure 7.10 shows a screen for testing performance of a Windows Form application. Table 7.1 shows the machine configurations for the test environment. Table 7.2 shows the algorithm used for the test. As can be seen from the Figure 7.11, there is no significant difference in running the performance test in ASP.NET pages or in Windows Form Application in a production environment. Running the ASP.NET pages in the Visual Studio.NET IDE shows a severe degradation in performance. What is most surprising is that the test results show no significant degradation in performance when run on a
workstation, away from the Web Service Web Server. This indicates a significant administrative usefulness of the EWSS system. The system can be run from anywhere in the network. In fact, the performance depends on the Host machine for the Web Service, rather than the Client workstation.

![WebForm1 - Microsoft Internet Explorer](http://localhost/EWSSaspCH/EWSSperf.aspx)

**Figure 7.9 ASP.NET Performance Test Screen for processing all of Form Five**
Figure 7.10 Windows Forms Performance Test Screen

<table>
<thead>
<tr>
<th>Environment</th>
<th>Average Time in seconds</th>
<th>Readings</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 ASP NET localhost, Machine B, run from IDE</td>
<td>88.75</td>
<td>102 83 84 86</td>
</tr>
<tr>
<td>12 ASP NET localhost, Machine B, run from Explorer</td>
<td>7.44</td>
<td>7.31 7.25 7.75 7.44</td>
</tr>
<tr>
<td>13 Windows Form App, localhost, Machine B, run from IDE</td>
<td>8.49</td>
<td>8.9 25 8.31 8.41</td>
</tr>
<tr>
<td>14 Windows Form App, localhost, Machine B, run as exe</td>
<td>7.47</td>
<td>7.81 7.41 7.19 7.47</td>
</tr>
<tr>
<td>15 ASP NET workstation, Machine C</td>
<td>7.4</td>
<td>7.87 7.25 7.46 7.03</td>
</tr>
<tr>
<td>16 Windows Form App, workstation, Machine C</td>
<td>7.77</td>
<td>8.03 7.66 7.69 7.69</td>
</tr>
</tbody>
</table>

Figure 7.11 Performance of ASP.NET and Windows Forms Application