Testing

Testing is a very big and important topic when it comes to software development.

Testing has a number of aspects that need to be considered.

System stability – is the system going to crash or not?
System usability – is the system easy to use?
System security – is the data and code safe from hackers?

Testing will never be the most exciting task in the world however it is one of the most important when it comes to programming.

Errors are good!

The first thing that I want to say about testing is that errors are good.

I do not know a single person who doesn't make mistakes especially mistakes when creating computer based applications.

If you create a system and tell me that you tested it and found no problems I simply won't believe you.

There is always something wrong with your work at some point in the design.

If it isn't an actual fault then there will always be something that could have been done better.

We test to find errors and they will exist, therefore we need to find them!

In documenting your testing I expect you to state clearly what errors you found and what you did about them in your test logs.

If you have tests with no errors and no resulting changes to your system, your test strategy is simply flawed.

When to Test

Testing that is carried out prior to the release is called formative testing. Testing carried out after the site is complete is called summative testing.

Formative testing directs the design of the site as it is in development. It guides the design process to keep it on track.

Summative testing takes place after the project is live and could be the source of some embarrassment as problems that should have been found at an earlier stage come to light.

Summative testing does have a place should one go to the trouble of evaluating similar sites already out there.

Imagine a chef making soup.

The formative stage is the chef tasting the soup as they make it.

The summative phase is when the customer tastes it.

In the design process described in these lectures testing should take place at all stages of the design process.

Problems that are identified early on are much less costly than problems found after the site has gone live.

Why do we test?

The most obvious reason to test your site is to find faults or errors with the code. Bugs need resolving before the application goes live.

However, because the suitability of our systems may be open to interpretation we need to be very careful that our design is suitable for the user.

You may create an interface that looks perfectly fine to you as the developer however it fails when placed in front of the customer.

Why is this?

One problem is that as the author of your system you know it back to front. You know where everything is and what buttons to press to perform different tasks. You will inevitably be an expert on your own design and as a result find it hard to imagine what it is like to approach your design for the first time.

We need to have our designs viewed through fresh eyes to see the things that are hard to see ourselves.

What do we test?

As stated above, we need to test the code that makes up our system however we also need to test other aspects of the system that may be harder to identify than a broken link.

We need to find some way of establishing the following point.

- Does our system provide "ease of use"?

For the sake of this lecture I shall split testing into two types.

- System testing
- Usability testing

System Testing

There are a number of issues that need to be considered under the heading of system testing.

You will need to consider issues such as the following.

- What happens when bad data is entered?
- Do all of the links to pages work?
- Does your design work on different browsers?
- Does your design work at different screen resolutions?
- How does your design behave on different computer platforms?

Is your code correct?

Black Box Testing

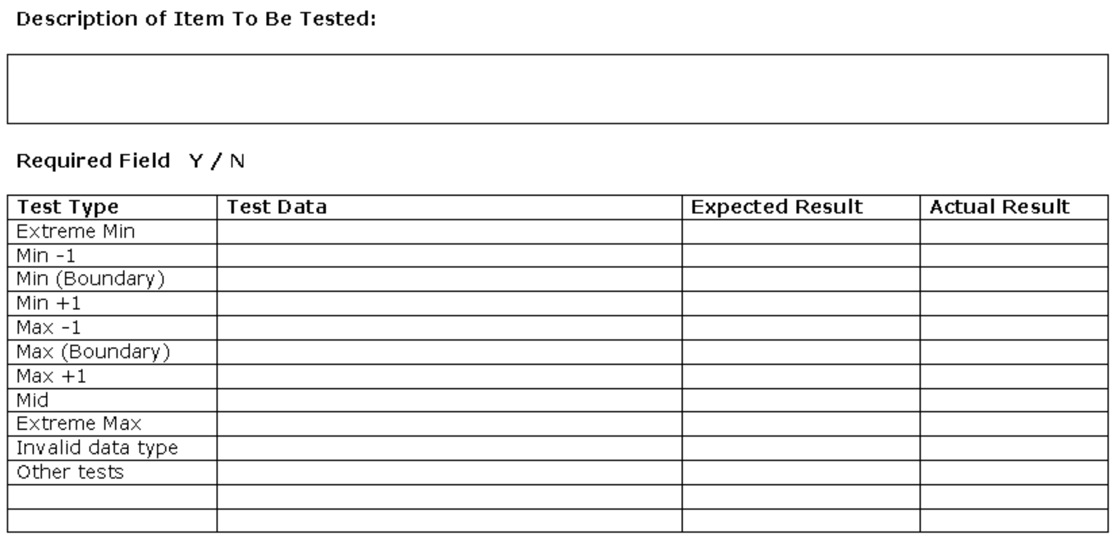
In the lab last week you will have created the query in the data layer and set up the web form to enter data. We will not at this stage have any code that makes the link between the two layers.

Even though we have no code in place we may still create tests that may be applied as soon as the system is finished. This approach to testing sees the system as a black box. We don't care how the code works, but we do know what values should / shouldn't be accepted even at the early stages.

Completing the Test Plan & Log

You have been provided with a standard format for test plans / logs which you need to learn how to complete.

Below is the layout for the test plan & log…



Description of Item To Be Tested:

Required Field Y / N

Test Type	Test Data	Expected Result	Actual Result
Extreme Min			
Min -1			
Min (Boundary)			
Min +1			
Max -1			
Max (Boundary)			
Max +1			
Mid			
Extreme Max			
Invalid data type			
Other tests			

The pro forma is available for download from the main web site.

The following example will assume that there is a field on a web form for entering the age of a staff member.

Description of Item to be Tested

This should contain a brief description of the field in question under test.

e.g.

“Age field for staff member.”

Extreme Min(imum)

The Extreme Min is an example of an excessively small value for the field. This may never be input under normal circumstances however it is important to test your application for unexpected situations.

An extreme minimum in the case of a person's age might be any value in minus figures e.g. -300

Test Type	**Test Data**
Extreme Min	-300
Min -1	
Min (Boundary)	
Min +1	
Max -1	
Max (Boundary)	
Max +1	
Mid	
Extreme Max	
Invalid data type	
Other tests	

Min (Boundary)

The Min (Boundary) test data is the smallest acceptable value for the field.

In the case of the age the smallest acceptable value may be zero, or it may be another value such as 16 depending on the rules for the circumstance.

In this case we shall assume that the smallest acceptable value is 13 as this is the youngest age legally allowable. Based on this we may generate test data for the Min -1 and Min +1 values.

Test Type	Test Data
Extreme Min	-300
Min -1	12
Min (Boundary)	13
Min +1	14
Max -1	
Max (Boundary)	
Max +1	
Mid	
Extreme Max	
Invalid data type	
Other tests	

This is referred to as a boundary test as it tests the minimum acceptable value and either side of that boundary.

Max (Boundary)

The Max (Boundary) is the maximum allowable value for the field. We could specify some arbitrary figure such as 150 however there is such a thing as a retirement age (currently 65) however workers do have an option to work for longer, in this case we shall set the age max to a semi arbitrary 80.

Test Type	Test Data
Extreme Min	-300
Min -1	12
Min (Boundary)	13
Min +1	14
Max -1	79
Max (Boundary)	80
Max +1	81
Mid	
Extreme Max	
Invalid data type	
Other tests	

Again this allows us to generate the -1 and +1 values.

Mid

The mid range value is a value half way between the Max and Min boundary values. Min = 13 Max = 80 therefore Mid = 47.

Test Type	**Test Data**
Extreme Min	-300
Min -1	12
Min (Boundary)	13
Min +1	14
Max -1	79
Max (Boundary)	80
Max +1	81
Mid	47
Extreme Max	
Invalid data type	
Other tests	

Extreme Max

This is an extremely large value that hopefully may never occur e.g. +300

Test Type	**Test Data**
Extreme Min	-300
Min -1	12
Min (Boundary)	13
Min +1	14
Max -1	79
Max (Boundary)	80
Max +1	81
Mid	47
Extreme Max	300
Invalid data type	
Other tests	

Invalid Data Type

In this example of age, the data type is an integer value. We need to generate a few values which are inappropriate for the integer data type e.g.

String	“Fred”
Date	01/01/2001
Decimal	14.5

Test Type	**Test Data**
Extreme Min	-300
Min -1	12
Min (Boundary)	13
Min +1	14
Max -1	79
Max (Boundary)	80
Max +1	81
Mid	47
Extreme Max	300
Invalid data type	Fred
	01/01/2001
	14.5
Other tests	Anything else you can think of

Other Tests

This relates to any other testing that you can possibly think of. Should the field be a required field then it will be a good idea to test what happens should the field be left blank.

Once the test data for that field has been compiled, one should then move on to other fields within the program.

Once all of the test data has been assembled the tests should be run and the test plan & log annotated indicating the results of the test.

Should any test conditions not run as expected then the developer must be notified in order to debug and correct the code.

Prototyping

Another useful approach to testing our applications is to use a system of prototyping to assess the usability of our design.

Consider the following prototype…

A prototype aircraft might have the following features of the finished article:

- Shape
- Proportion
- Colour
- Wings

The prototype may not have

- Glass windows
- Engines or any internal systems

It almost certainly won't have

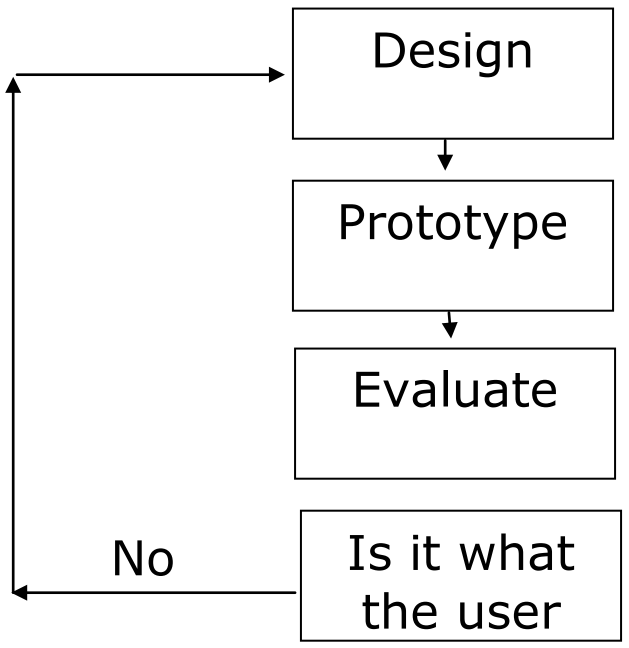
- Power of flight (due to lack of internal systems)
- Ability to carry passengers (due to the fact that the prototype is only 1 meter long!)

Prototyping Life Cycle

One important question when it comes to usability is this…

How do you know if the design for my application is correct?

By seeking input from the client and potential users at each stage of the design process we may answer this question.



But what exactly is Usability?

Usability may be defined by the following questions.

- How easy is the application to learn?
- How quickly are tasks performed?
- How easy is it to remember how to use the application?
- How many errors does a user make?
- Is the user satisfied with the experience?

Who to use?

You need to locate potential users of the site and obtain their feedback.

How many test subjects?

It has been suggest by Jakob Nielsen that you only need five test subjects. The argument states that after five users you will end up watching the same mistakes being made over and over again.

To find out more visit the following page.

http://www.useit.com/alertbox/20000319.html

His argument is based on a position of cost versus benefits.

One point however is that testing zero users gives you zero insight.

Approaches to Testing

Focus Groups

A focus group may be of value in comparing two different approaches to the design.

"Would this site be better structured using a hub and spoke navigation model or a linear one?"

Testing the Paper Based Design

This test would involve potential end users being given a paper based design and allowing them to pretend it was on the computer.

How would they imagine the system would work?
What changes would they make to the site?

The test would be observed and notes taken by the subjects and the observer.

Card Sorting

May be used to identify an organisational scheme.

Write down on file cards the names of the sections of your site with a short description.

Give the cards to different potential users and ask then to sort the cards in a way that makes sense to them.

For more details follow the link below.

http://www.useit.com/alertbox/20040719.html

Design an Experiment

In this case potential users are given a task to perform.

For example locate a product, add it to the cart and then proceed to checkout.

The task is observed, timed and both observer and subject make notes on the performance.

Planning Test Scenarios

It isn't good enough to simply ask a user to "find something and see how you get on".

For a test to provide worthwhile data a test scenario is required.

For example

Find a pair of men's trousers in black for less than £25.

Make sure that the user starts at the home page and all data is in an initialised state.

Screen recording software (CamStudio) could be used to record the test and observe the results at a later date.

A web cam could be used to observe the users expression.

Avoid Polluting the Test

One problem you must take into account is that the behaviour of a user may change when they know that they are being observed.

There is evidence to suggest that people work better because they are under test conditions.

The tests should try to be as much like real situations as possible.

This obviously means that if a person gets stuck using your web site you must not jump in and help them!

White Box Testing

The good thing about the test strategies above is that it is possible to perform a great deal of testing without ever having any working code.

At some point though we are going to have a system nearly ready for release and we will need to test not just individual classes but also individual lines of code.

To do this testing we often write specific programs that do the testing for us.

Imagine for example we have a class that controls a customer’s bank balance.

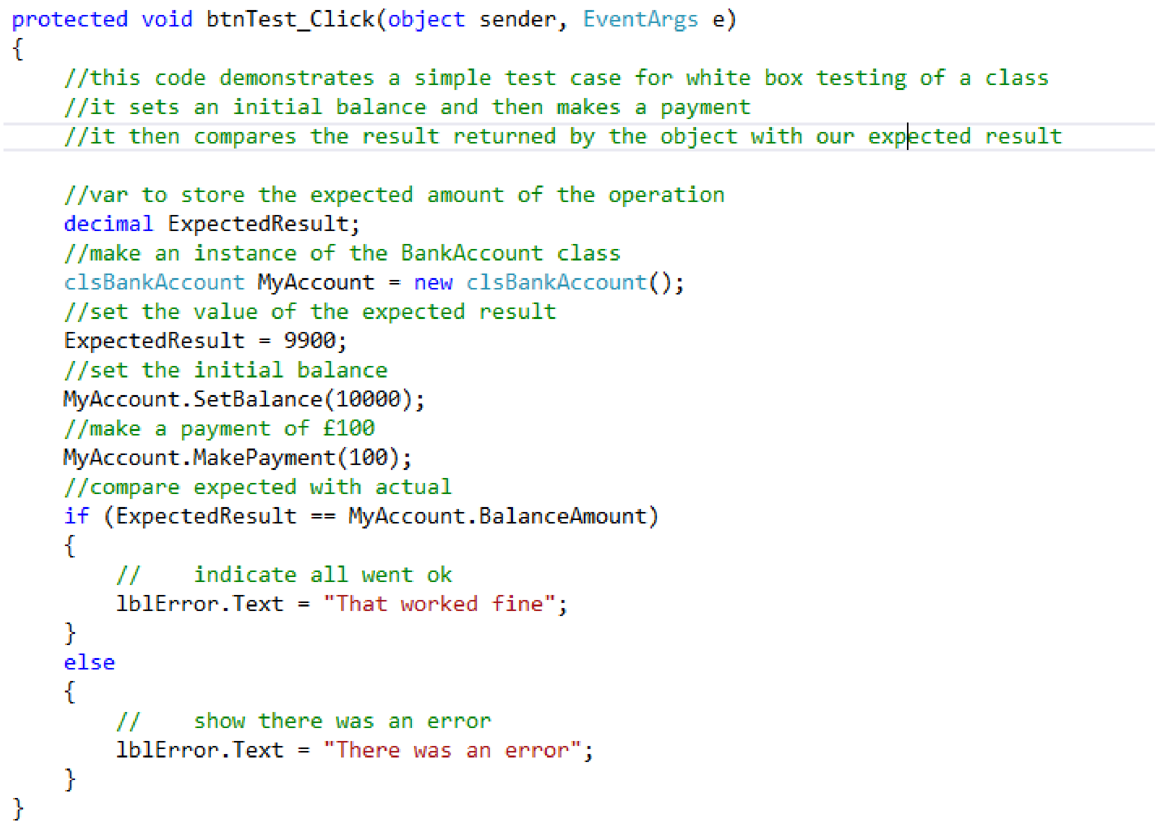
The class has the following methods…

```
SetBalance(BalanceAmount as decimal)
MakePayment(PaymentAmount as decimal)
```

The class also has the following property…

```
BalanceAmount
```

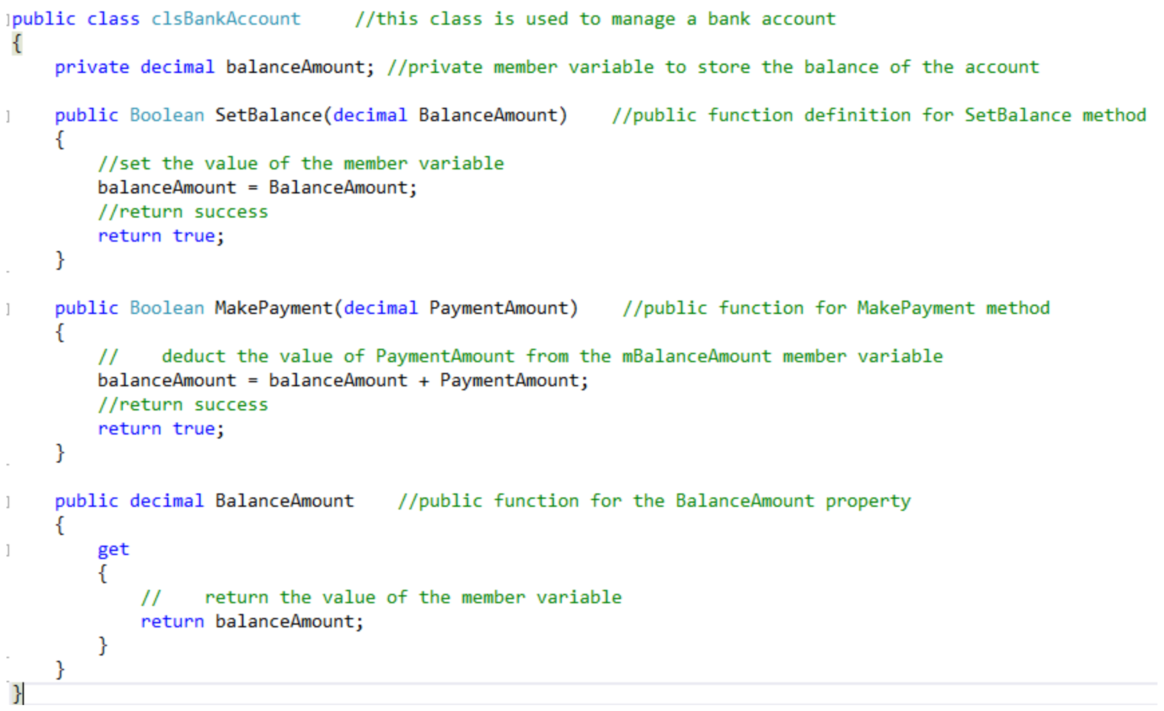
Take a look at the following test code…



```
protected void btnTest_Click(object sender, EventArgs e)
{
    //this code demonstrates a simple test case for white box testing of a class
    //it sets an initial balance and then makes a payment
    //it then compares the result returned by the object with our expected result

    //var to store the expected amount of the operation
    decimal ExpectedResult;
    //make an instance of the BankAccount class
    clsBankAccount MyAccount = new clsBankAccount();
    //set the value of the expected result
    ExpectedResult = 9900;
    //set the initial balance
    MyAccount.SetBalance(10000);
    //make a payment of £100
    MyAccount.MakePayment(100);
    //compare expected with actual
    if (ExpectedResult == MyAccount.BalanceAmount)
    {
        //    indicate all went ok
        lblError.Text = "That worked fine";
    }
    else
    {
        //    show there was an error
        lblError.Text = "There was an error";
    }
}
```

This is also a good opportunity to take a look inside our own custom class to see exactly how it is constructed…



```
public class clsBankAccount     //this class is used to manage a bank account
{
    private decimal balanceAmount; //private member variable to store the balance of the account

    public Boolean SetBalance(decimal BalanceAmount)    //public function definition for SetBalance method
    {
        //set the value of the member variable
        balanceAmount = BalanceAmount;
        //return success
        return true;
    }

    public Boolean MakePayment(decimal PaymentAmount)    //public function for MakePayment method
    {
        //    deduct the value of PaymentAmount from the mBalanceAmount member variable
        balanceAmount = balanceAmount + PaymentAmount;
        //return success
        return true;
    }

    public decimal BalanceAmount    //public function for the BalanceAmount property
    {
        get
        {
            //    return the value of the member variable
            return balanceAmount;
        }
    }
}
```

(The code for this application is available from the module web site.)