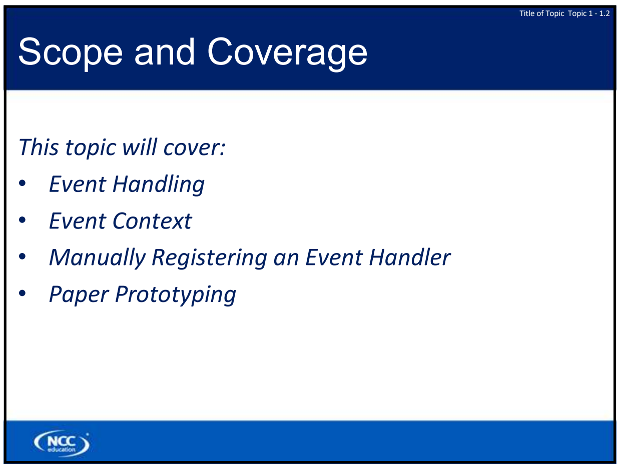
Awarding Great British Qualifications

Designing and Developing Object-Oriented Computer Programmes
Topic 2: Event Driven Programming

1

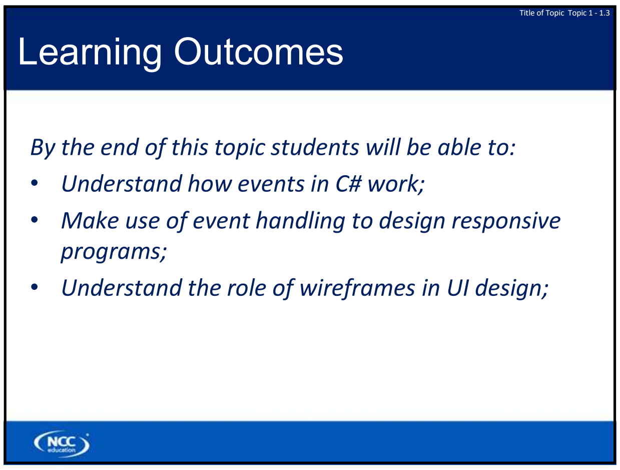


Title of Topic Topic 1 - 1.2

Scope and Coverage

This topic will cover:
- *Event Handling*
- *Event Context*
- *Manually Registering an Event Handler*
- *Paper Prototyping*

2

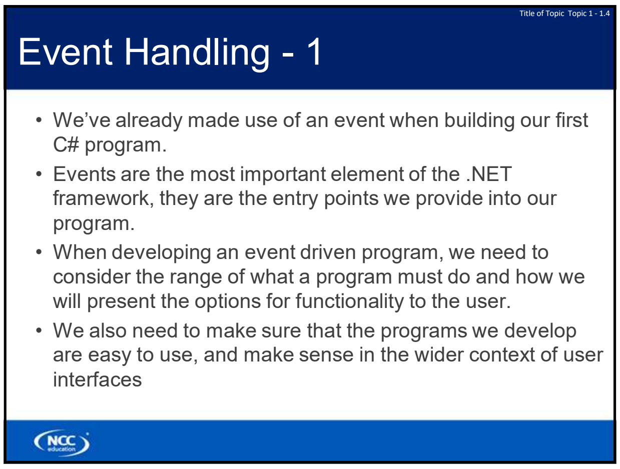


Title of Topic Topic 1 - 1.3

Learning Outcomes

By the end of this topic students will be able to:
- *Understand how events in C# work;*
- *Make use of event handling to design responsive programs;*
- *Understand the role of wireframes in UI design;*

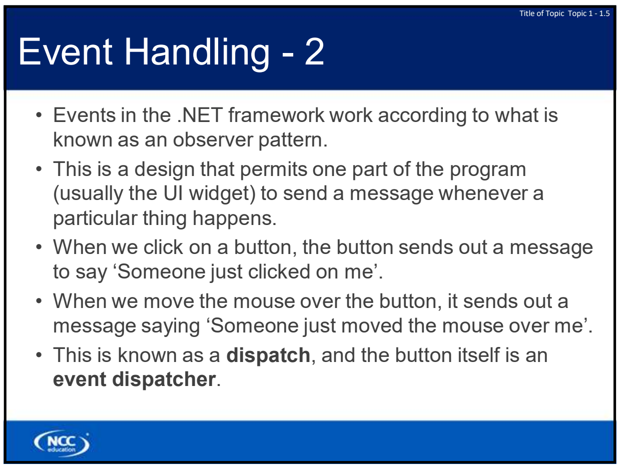
3



Event Handling - 1

- We’ve already made use of an event when building our first C# program.
- Events are the most important element of the .NET framework, they are the entry points we provide into our program.
- When developing an event driven program, we need to consider the range of what a program must do and how we will present the options for functionality to the user.
- We also need to make sure that the programs we develop are easy to use, and make sense in the wider context of user interfaces

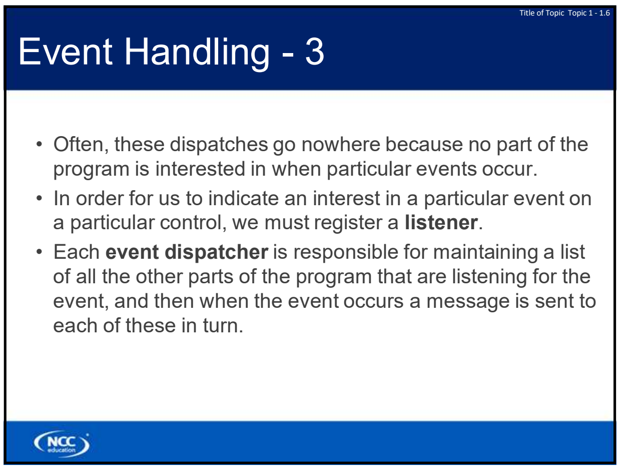
4



Event Handling - 2

- Events in the .NET framework work according to what is known as an observer pattern.
- This is a design that permits one part of the program (usually the UI widget) to send a message whenever a particular thing happens.
- When we click on a button, the button sends out a message to say ‘Someone just clicked on me’.
- When we move the mouse over the button, it sends out a message saying ‘Someone just moved the mouse over me’.
- This is known as a **dispatch**, and the button itself is an **event dispatcher**.

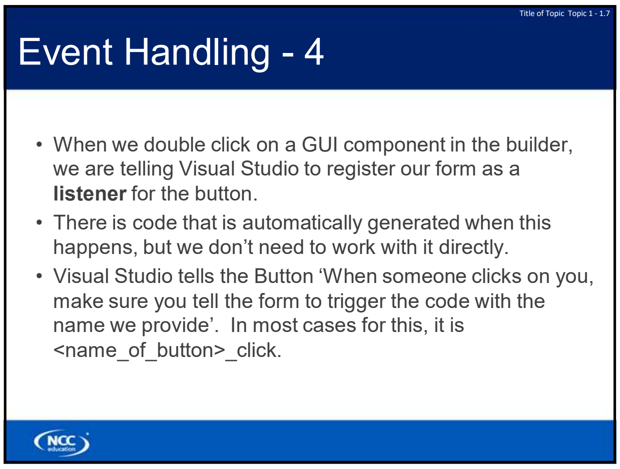
5



Event Handling - 3

- Often, these dispatches go nowhere because no part of the program is interested in when particular events occur.
- In order for us to indicate an interest in a particular event on a particular control, we must register a **listener**.
- Each **event dispatcher** is responsible for maintaining a list of all the other parts of the program that are listening for the event, and then when the event occurs a message is sent to each of these in turn.

6

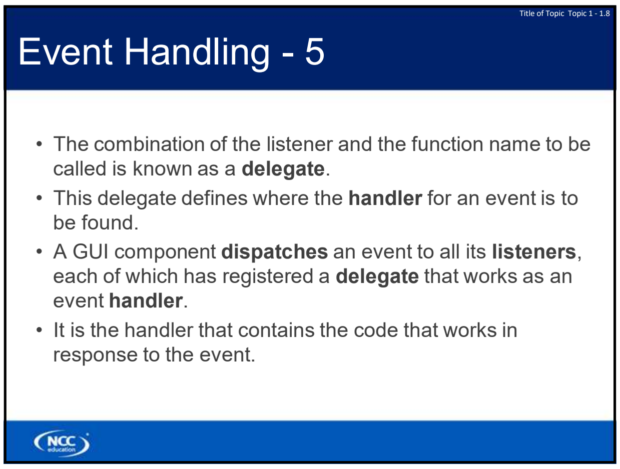


Title of Topic Topic 1 - 1.7

Event Handling - 4

- When we double click on a GUI component in the builder, we are telling Visual Studio to register our form as a **listener** for the button.
- There is code that is automatically generated when this happens, but we don't need to work with it directly.
- Visual Studio tells the Button 'When someone clicks on you, make sure you tell the form to trigger the code with the name we provide'. In most cases for this, it is <name_of_button>_click.

7

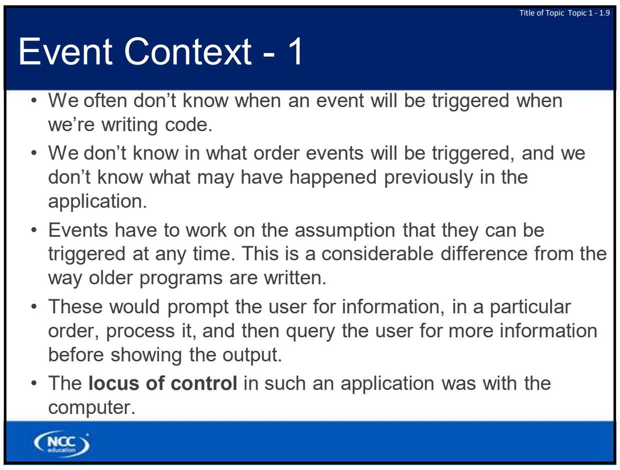


Title of Topic Topic 1 - 1.8

Event Handling - 5

- The combination of the listener and the function name to be called is known as a **delegate**.
- This delegate defines where the **handler** for an event is to be found.
- A GUI component **dispatches** an event to all its **listeners**, each of which has registered a **delegate** that works as an event **handler**.
- It is the handler that contains the code that works in response to the event.

8

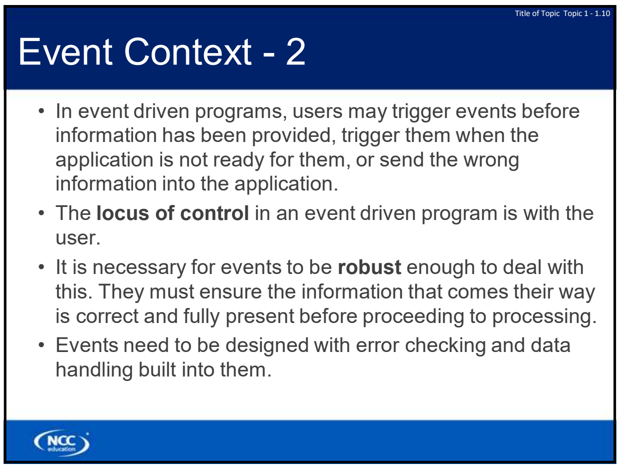


Title of Topic Topic 1 - 1.9

Event Context - 1

- We often don't know when an event will be triggered when we're writing code.
- We don't know in what order events will be triggered, and we don't know what may have happened previously in the application.
- Events have to work on the assumption that they can be triggered at any time. This is a considerable difference from the way older programs are written.
- These would prompt the user for information, in a particular order, process it, and then query the user for more information before showing the output.
- The **locus of control** in such an application was with the computer.

9

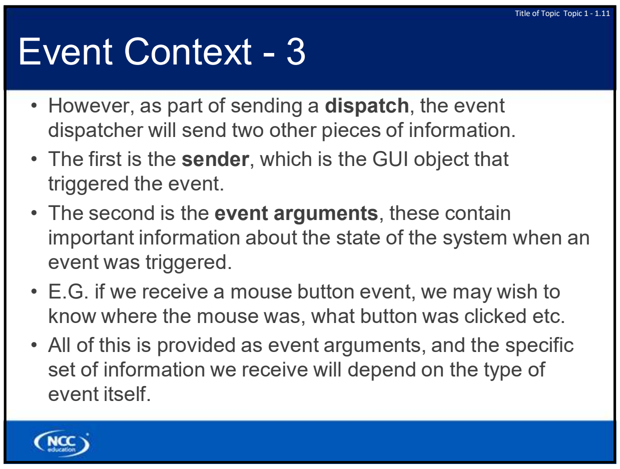


Title of Topic Topic 1 - 1.10

Event Context - 2

- In event driven programs, users may trigger events before information has been provided, trigger them when the application is not ready for them, or send the wrong information into the application.
- The **locus of control** in an event driven program is with the user.
- It is necessary for events to be **robust** enough to deal with this. They must ensure the information that comes their way is correct and fully present before proceeding to processing.
- Events need to be designed with error checking and data handling built into them.

10

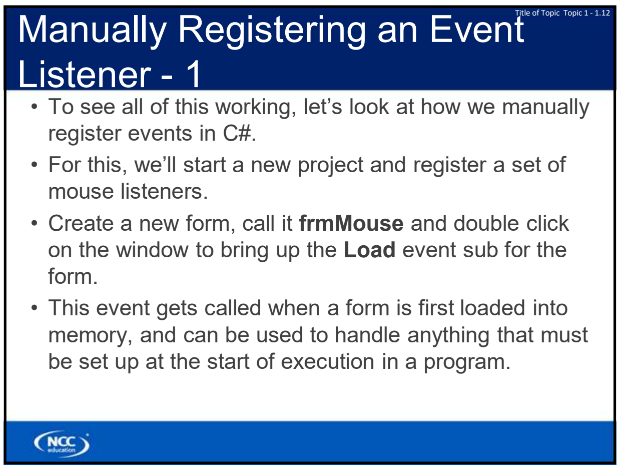


Title of Topic Topic 1 - 1.11

Event Context - 3

- However, as part of sending a **dispatch**, the event dispatcher will send two other pieces of information.
- The first is the **sender**, which is the GUI object that triggered the event.
- The second is the **event arguments**, these contain important information about the state of the system when an event was triggered.
- E.G. if we receive a mouse button event, we may wish to know where the mouse was, what button was clicked etc.
- All of this is provided as event arguments, and the specific set of information we receive will depend on the type of event itself.

11

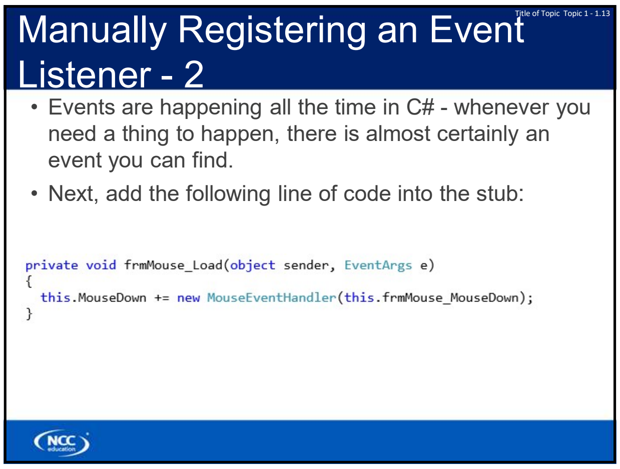


Title of Topic Topic 1 - 1.12

Manually Registering an Event Listener - 1

- To see all of this working, let’s look at how we manually register events in C#.
- For this, we’ll start a new project and register a set of mouse listeners.
- Create a new form, call it **frmMouse** and double click on the window to bring up the **Load** event sub for the form.
- This event gets called when a form is first loaded into memory, and can be used to handle anything that must be set up at the start of execution in a program.

12



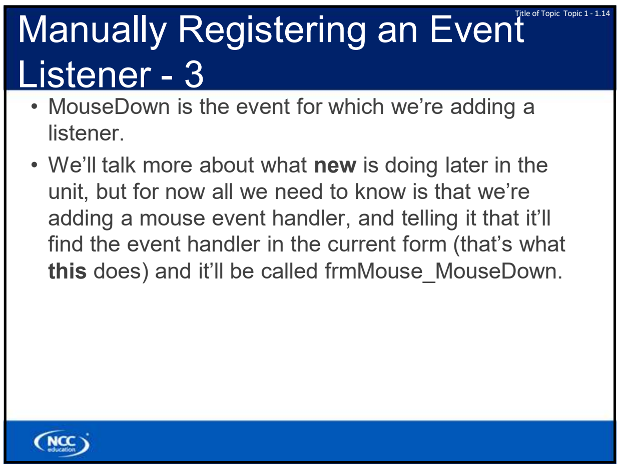
Title of Topic Topic 1 - 1.13

Manually Registering an Event Listener - 2

- Events are happening all the time in C# - whenever you need a thing to happen, there is almost certainly an event you can find.
- Next, add the following line of code into the stub:

```
private void frmMouse_Load(object sender, EventArgs e)
{
  this.MouseDown += new MouseEventHandler(this.frmMouse_MouseDown);
}
```

13

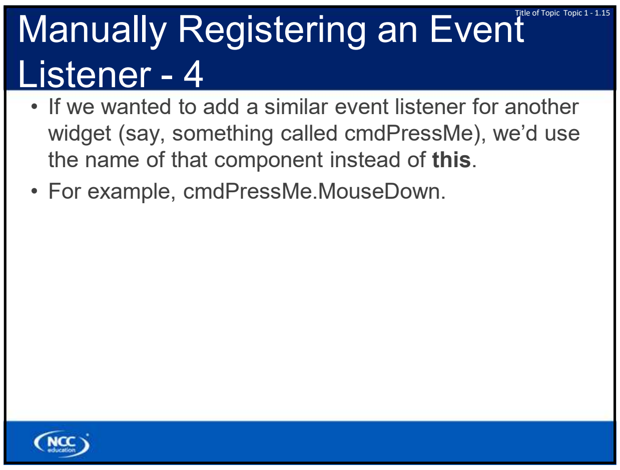


Title of Topic Topic 1 - 1.14

Manually Registering an Event Listener - 3

- MouseDown is the event for which we're adding a listener.
- We'll talk more about what **new** is doing later in the unit, but for now all we need to know is that we're adding a mouse event handler, and telling it that it'll find the event handler in the current form (that's what **this** does) and it'll be called frmMouse_MouseDown.

14

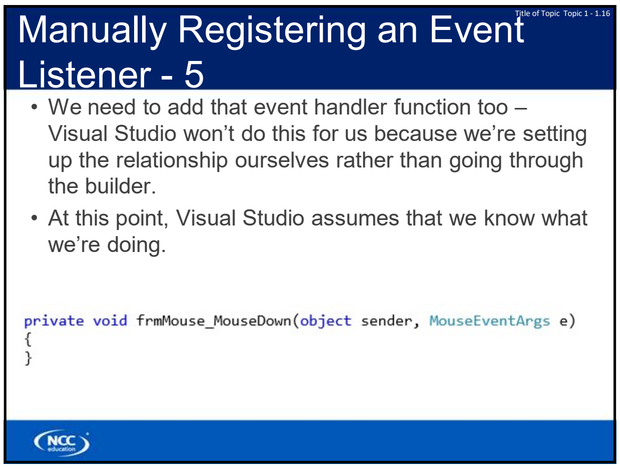


Title of Topic Topic 1 - 1.15

Manually Registering an Event Listener - 4

- If we wanted to add a similar event listener for another widget (say, something called cmdPressMe), we'd use the name of that component instead of **this**.
- For example, cmdPressMe.MouseDown.

15

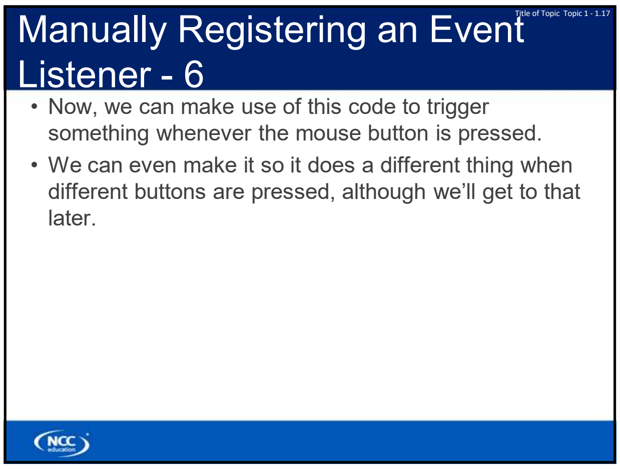


Manually Registering an Event Listener - 5

- We need to add that event handler function too – Visual Studio won't do this for us because we're setting up the relationship ourselves rather than going through the builder.
- At this point, Visual Studio assumes that we know what we're doing.

```
private void frmMouse_MouseDown(object sender, MouseEventArgs e)
{
}
```

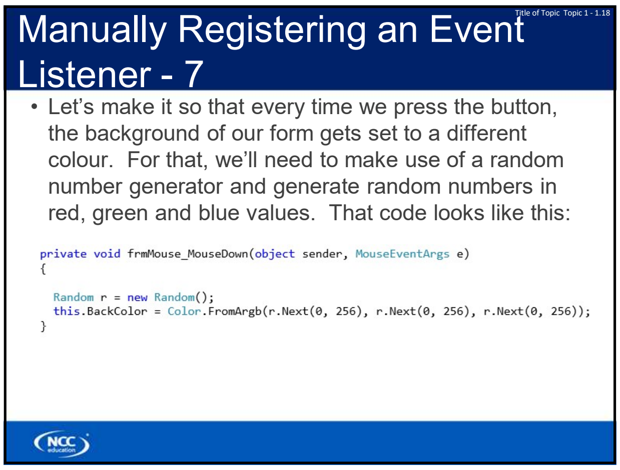
16



Manually Registering an Event Listener - 6

- Now, we can make use of this code to trigger something whenever the mouse button is pressed.
- We can even make it so it does a different thing when different buttons are pressed, although we'll get to that later.

17

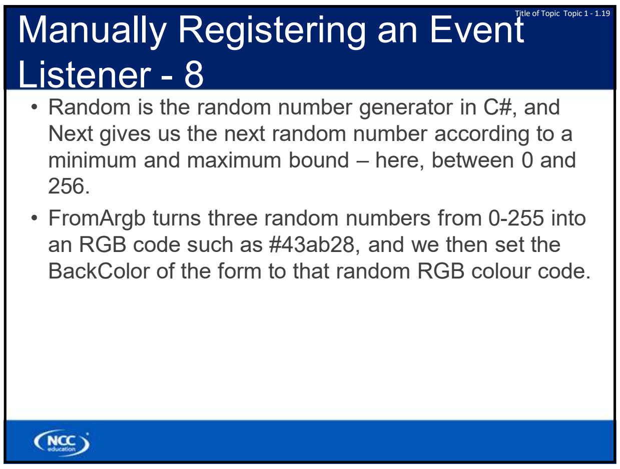


Manually Registering an Event Listener - 7

- Let's make it so that every time we press the button, the background of our form gets set to a different colour. For that, we'll need to make use of a random number generator and generate random numbers in red, green and blue values. That code looks like this:

```
private void frmMouse_MouseDown(object sender, MouseEventArgs e)
{

  Random r = new Random();
  this.BackColor = Color.FromArgb(r.Next(0, 256), r.Next(0, 256), r.Next(0, 256));
}
```

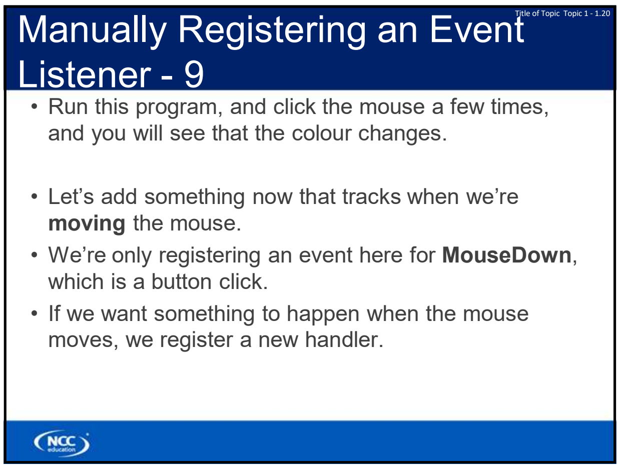
18



Manually Registering an Event Listener - 8

- Random is the random number generator in C#, and Next gives us the next random number according to a minimum and maximum bound – here, between 0 and 256.
- FromArgb turns three random numbers from 0-255 into an RGB code such as #43ab28, and we then set the BackColor of the form to that random RGB colour code.

19

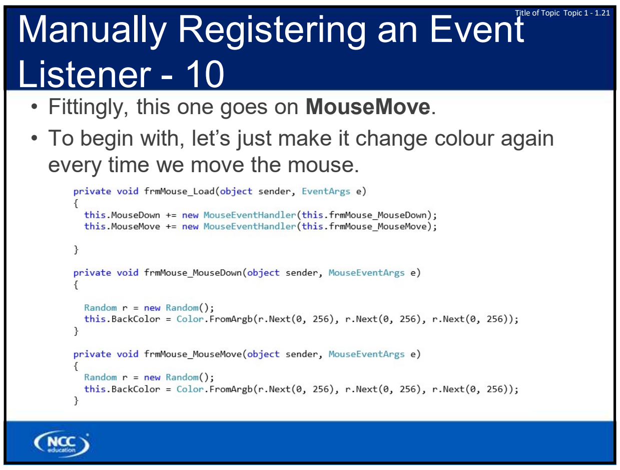


Manually Registering an Event Listener - 9

- Run this program, and click the mouse a few times, and you will see that the colour changes.

- Let's add something now that tracks when we're **moving** the mouse.
- We're only registering an event here for **MouseDown**, which is a button click.
- If we want something to happen when the mouse moves, we register a new handler.

20

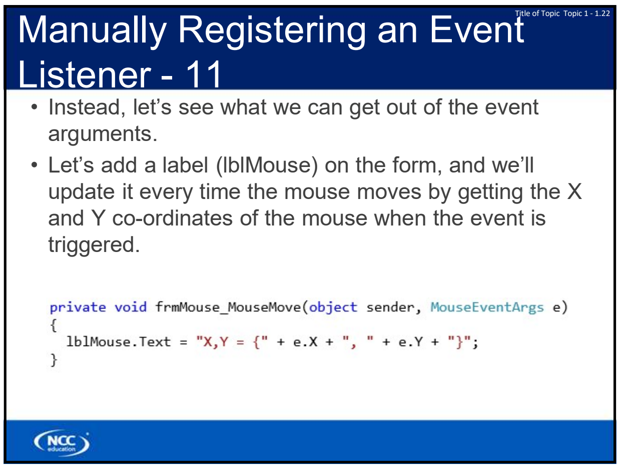


Manually Registering an Event Listener - 10

- Fittingly, this one goes on **MouseMove**.
- To begin with, let's just make it change colour again every time we move the mouse.

```
private void frmMouse_Load(object sender, EventArgs e)
{
  this.MouseDown += new MouseEventHandler(this.frmMouse_MouseDown);
  this.MouseMove += new MouseEventHandler(this.frmMouse_MouseMove);

}

private void frmMouse_MouseDown(object sender, MouseEventArgs e)
{

  Random r = new Random();
  this.BackColor = Color.FromArgb(r.Next(0, 256), r.Next(0, 256), r.Next(0, 256));
}

private void frmMouse_MouseMove(object sender, MouseEventArgs e)
{
  Random r = new Random();
  this.BackColor = Color.FromArgb(r.Next(0, 256), r.Next(0, 256), r.Next(0, 256));
}
```

21



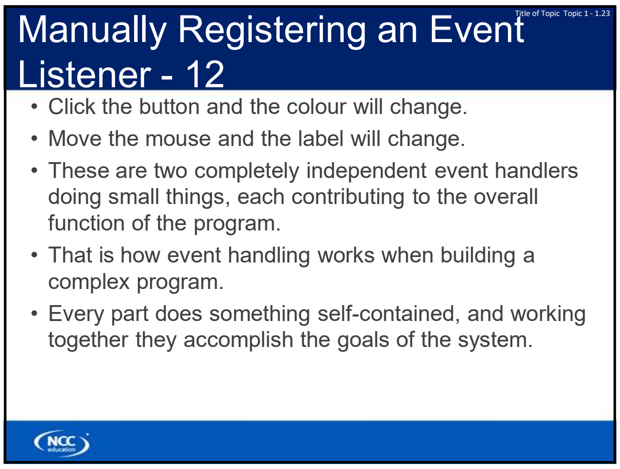
Title of Topic Topic 1 - 1.22

Manually Registering an Event Listener - 11

- Instead, let's see what we can get out of the event arguments.
- Let's add a label (lblMouse) on the form, and we'll update it every time the mouse moves by getting the X and Y co-ordinates of the mouse when the event is triggered.

```
private void frmMouse_MouseMove(object sender, MouseEventArgs e)
{
  lblMouse.Text = "X,Y = {" + e.X + ", " + e.Y + "}";
}
```

22

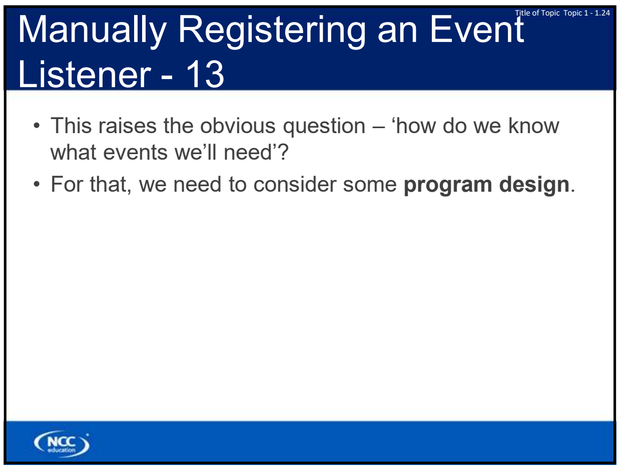


Title of Topic Topic 1 - 1.23

Manually Registering an Event Listener - 12

- Click the button and the colour will change.
- Move the mouse and the label will change.
- These are two completely independent event handlers doing small things, each contributing to the overall function of the program.
- That is how event handling works when building a complex program.
- Every part does something self-contained, and working together they accomplish the goals of the system.

23

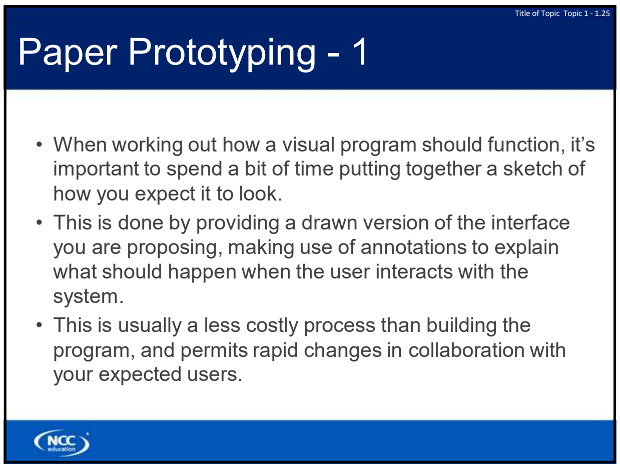


Title of Topic Topic 1 - 1.24

Manually Registering an Event Listener - 13

- This raises the obvious question – 'how do we know what events we'll need'?
- For that, we need to consider some **program design**.

24



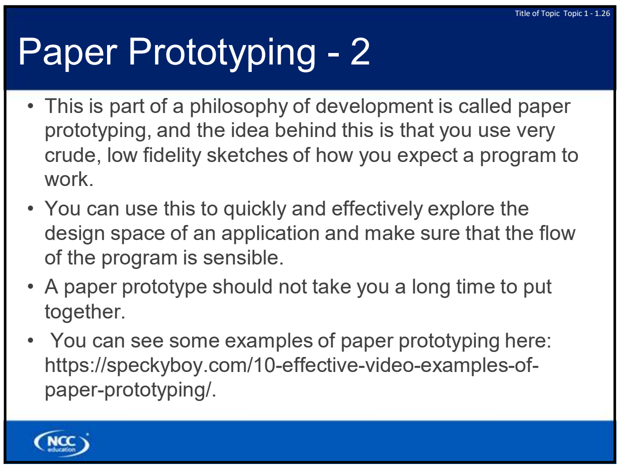
Title of Topic Topic 1 - 1.25

Paper Prototyping - 1

- When working out how a visual program should function, it's important to spend a bit of time putting together a sketch of how you expect it to look.
- This is done by providing a drawn version of the interface you are proposing, making use of annotations to explain what should happen when the user interacts with the system.
- This is usually a less costly process than building the program, and permits rapid changes in collaboration with your expected users.

NCC education

25



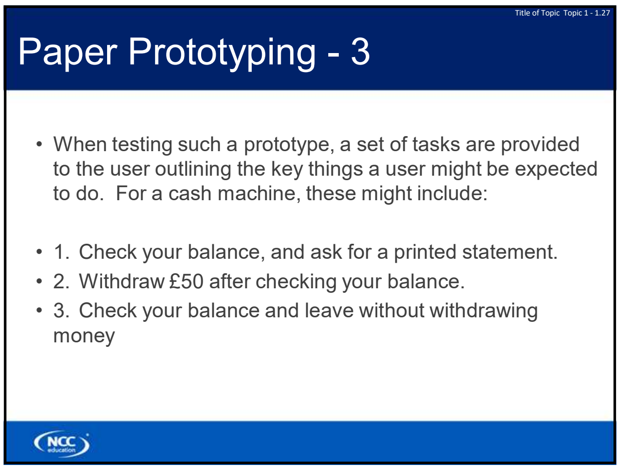
Title of Topic Topic 1 - 1.26

Paper Prototyping - 2

- This is part of a philosophy of development is called paper prototyping, and the idea behind this is that you use very crude, low fidelity sketches of how you expect a program to work.
- You can use this to quickly and effectively explore the design space of an application and make sure that the flow of the program is sensible.
- A paper prototype should not take you a long time to put together.
- You can see some examples of paper prototyping here: https://speckyboy.com/10-effective-video-examples-of-paper-prototyping/.

NCC education

26



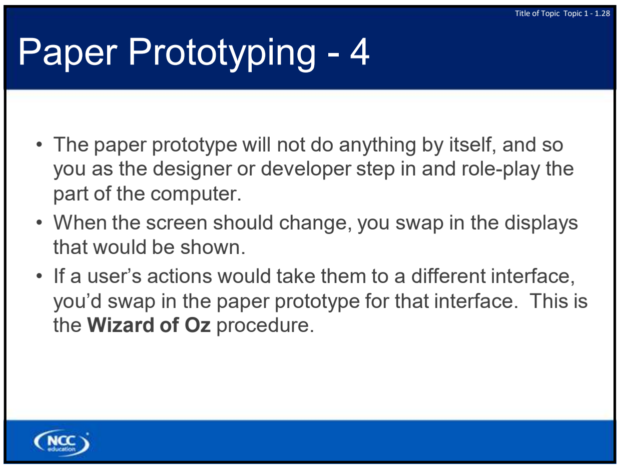
Title of Topic Topic 1 - 1.27

Paper Prototyping - 3

- When testing such a prototype, a set of tasks are provided to the user outlining the key things a user might be expected to do. For a cash machine, these might include:
- 1. Check your balance, and ask for a printed statement.
- 2. Withdraw £50 after checking your balance.
- 3. Check your balance and leave without withdrawing money

NCC education

27

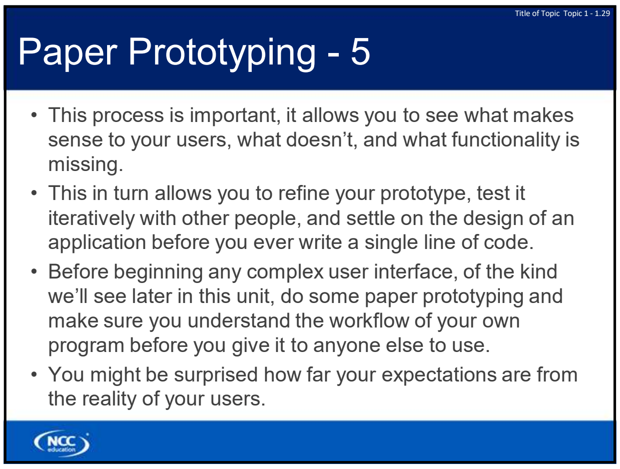


Title of Topic Topic 1 - 1.28

Paper Prototyping - 4

- The paper prototype will not do anything by itself, and so you as the designer or developer step in and role-play the part of the computer.
- When the screen should change, you swap in the displays that would be shown.
- If a user's actions would take them to a different interface, you'd swap in the paper prototype for that interface. This is the **Wizard of Oz** procedure.

28

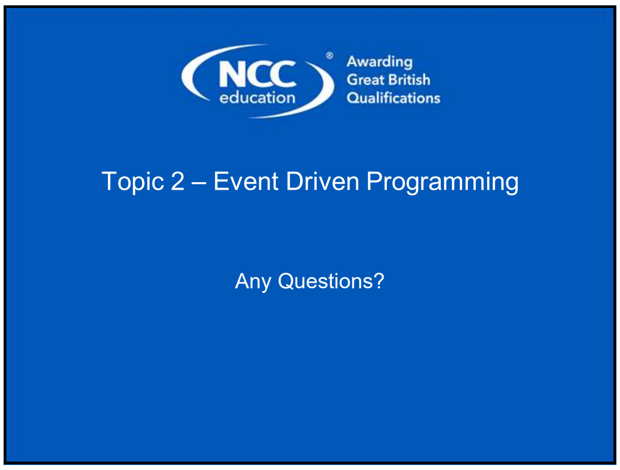


Title of Topic Topic 1 - 1.29

Paper Prototyping - 5

- This process is important, it allows you to see what makes sense to your users, what doesn't, and what functionality is missing.
- This in turn allows you to refine your prototype, test it iteratively with other people, and settle on the design of an application before you ever write a single line of code.
- Before beginning any complex user interface, of the kind we'll see later in this unit, do some paper prototyping and make sure you understand the workflow of your own program before you give it to anyone else to use.
- You might be surprised how far your expectations are from the reality of your users.

29



Awarding Great British Qualifications

Topic 2 – Event Driven Programming

Any Questions?

30