50002 - Software Engineering Design - Lecture 12

Oliver Killane

22/11/21

Lecture Recording

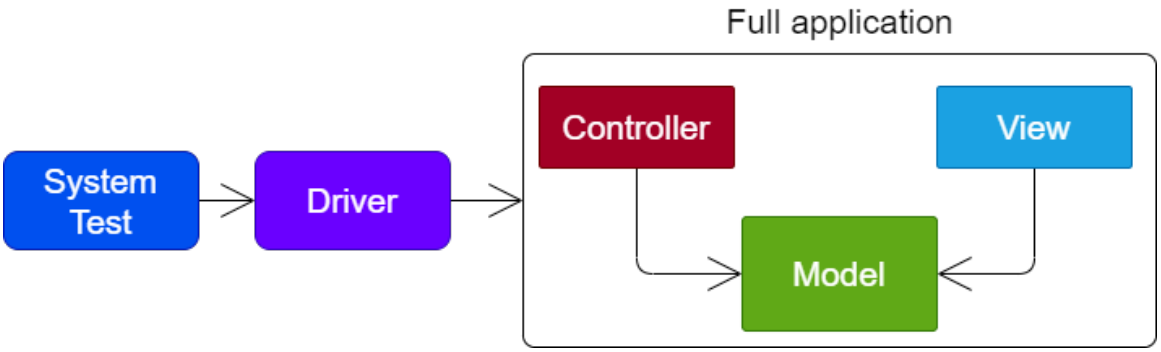
Lecture recording is available here

Testing

By using the **Model-View-Controller** pattern we can make it easy to test the logic of our application, as we can test the model in isolation.

However we may want to test the entire application to ensure not only that the view and controller components function, but that there are no bugs from integrating them together.

To do this we do a **System Test** which interacts with an instance of our application.



The driver interacts with the framework used for displaying graphics in order to get value of objects, and pass commands.

Window Licker

A driver used for testing Java based UIs, to use it we:

1. **Create driver code**
   To make it easier to access buttons, text firleds etc we can create functions to make use of WindowLick's library to get components and interact with them.

```
1  import ...
2
3  public class GUIDriver {
4      ...
5
6      // Helper function to click the button named "cool"
7      public void clickCoolButton() {
8          button("cool").click();
9      }
10
11     ...
12
13     // Helper function to get buttons with a given name attached.
14     //
```

```
    // In the creation of JComponents we should use .setName(name) to allow us
    // to identify them this way.
    /unchecked/
        private JButtonDriver(String name) {
        return new JButtonDriver(this, JButton.class, ComponentDriver.named(name
            ↪ ));
    }
}
```

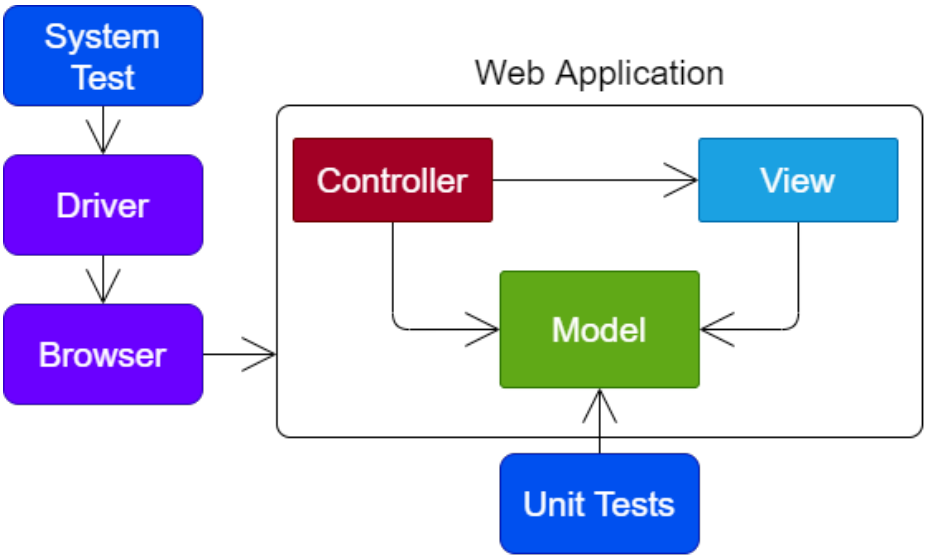
2. **Create Test Code**
   Using the driver to control, add test code.

```
import ...

public class GUITest {

    GUIDriver driver;

    @Before
    public void runApplication() {
        new GUI();
        driver = new GUIDriver();
    }

    @After
    public void StopApplication(){
        driver.closeApp();
    }

    @Test
    public void Test...

    ...
}
```

3. **Run Tests**
   Can run the tests in a headless mode, or display the GUI & let the **WindowLick** driver control interactions.

Web Application

We applications use the browser to view, and usually have the controller as the dominant object.



For webapp we can use a tool such as **Selenium** which automates browser control.

We can use **Selenium** to:

- connect to the webapp.
- Get handles for html components using tags, names, if necessary xpaths etc
- Interact with components.

It offers many drivers for interacting using different browsers, as well as headless modes, and can run many browsers in parallel to speed up large test suites.