



CHAPTER 11

How to Acceptance Test a User Interface



How to control a user interface

Suppose that the customer requests us to implement the following user story:

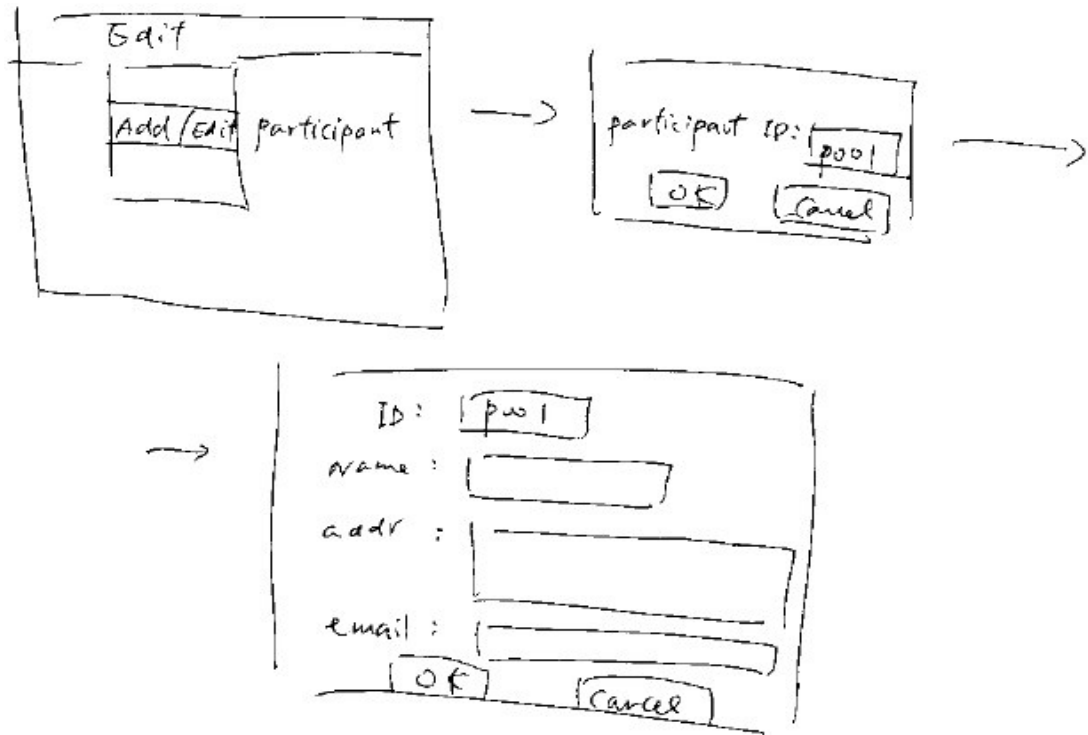
Name: Add or edit a participant

Events:

1. The user inputs a participant ID.
2. If it is a new ID, the user inputs the name, address and email of the participant.
3. If it is an existing ID, the system displays the name, address and email of the participant for the user to edit.
4. The system saves the information about the participant. In the future a user can input the ID of this participant, the system will retrieve the information about him.

Strictly speaking, the underlined portion in the above user story should not appear here because it describes the internal behavior of the system. What is important for the user is that he can retrieve the information of that participant in the future. However, we allow it here because it should help the user understand.

After discussing with us, the customer accepts our suggestion to adopt such a user interface: A user chooses a menu item "Add/Edit Participant" in the main window, then input a participant ID in a dialog. After clicking OK, the system will display another dialog to let the user input or edit the other information of the participant. After clicking OK, the system will save the information. There is a cancel button on both dialogs. Below is the sketches you drew on the white board during the discussion:



The question is, how to test this function that involves a user interface?

Test each user interface component separately

The above user story involves 3 user interface components:

1. The menu item "Add/Edit Participant".
2. The dialog allowing the user to input a participant ID. Let's call it ParticipantIdDialog.
3. The dialog allowing the user to input or edit the information of a participant. Let's call it ParticipantDetailsDialog.

Usually it is quite hard to test several user interface components together (Why? It takes quite some effort to explain. Therefore, please trust me. If not, you can try testing them all in one go and see). Therefore, usually we will test each user interface component separately.

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agileskills.org

Because the most complicated component in this example is ParticipantDetailsDialog, we will see how to test it first.

How to test ParticipantDetailsDialog

We can use the following test case to test ParticipantDetailsDialog: Initialize the system, provide a new participant ID to ParticipantDetailsDialog, input the name, address and email of the participant, click OK and see if the system has saved the information.

The corresponding test file may be:

```
SystemInit
ShowParticipantDetailsDialog,p001
SetParticipantName, Mike Chan
SetParticipantPassword, 888888
SetParticipantAddress, aaabbb
SetParticipantEmail, mike@excite.com
ClickOK
CheckParticipantStored,p001,888888, Mike Chan, aaabbb, mike@excite.com
```

In which the ShowParticipantDetailsDialog command creates a ParticipantDetailsDialog and provides it with p001 as the participant ID. The SetParticipantName command simulates the user's action of inputting the name of the participant (Mike Chan). Similarly, SetParticipantPassword, SetParticipantAddress and SetParticipantEmail simulate the user's action of inputting the password, address and email respectively. ClickOK simulates the action of clicking OK.

However, there is a problem here. For example, when SetParticipantName is run, how does it know which ParticipantDetailsDialog it should send the name "Mike Chan" to? In our system there is not a global ParticipantDetailsDialog object. Besides, as mentioned above, that ParticipantDetailsDialog is dynamically created by the ShowParticipantDetailsDialog command. Similarly, how does ClickOK know it should click the OK button on which dialog?

Therefore, we hope to use a variable in the test file to store that dialog, e.g.:

```
SystemInit
dialog=ShowParticipantDetailsDialog,p001
SetParticipantName,dialog, Mike Chan
SetParticipantPassword,dialog, 888888
SetParticipantAddress,dialog, aaabbb
SetParticipantEmail,dialog, mike@excite.com
ClickOK,dialog
CheckParticipantStored,p001,888888, Mike Chan, aaabbb, mike@excite.com
```

However, this will make the test files as complicated as a programming language. A possible solution is to make commands like SetParticipantName "sub-commands" of the ShowParticipantDetailsDialog command, e.g.:

```

SystemInit
ShowParticipantDetailsDialogStart,p001
    SetParticipantName,Mike Chan
    SetParticipantPassword,888888
    SetParticipantAddress,aaabbb
    SetParticipantEmail,mike@excite.com
    ClickOK
ShowParticipantDetailsDialogEnd
CheckParticipantStored,p001,888888,Mike Chan,aaabbb,mike@excite.com

```

In which the 7 lines from ShowParticipantDetailsDialogStart to ShowParticipantDetailsDialogEnd define only one ShowParticipantDetailsDialog command. Each of 5 lines in this command is a sub-command such as SetParticipantName. When this ShowParticipantDetailsDialog command is run, it will create a ParticipantDetailsDialog object and then tell all its 5 sub-commands to use this ParticipantDetailsDialog.

These commands can be implemented like this:

```

class ShowParticipantDetailsDialogCommand implements Command {
    String partId;
    SubCommand subCommands[];
    ShowParticipantDetailsDialogCommand(String partId) {
        this.partId=partId;
    }
    void addSubCommand(SubCommand subCommand) {
        //add subCommand to subCommands
        ...
    }
    boolean run() {
        ParticipantDetailsDialog partDetailsDlg=
            new ParticipantDetailsDialog(partId);
        //do not call showModal on the dialog, otherwise it will show on the screen,
        //waiting for a real user to interact with it.
        For (int i=0; i<subCommands.length; i++) {
            subCommands[i].setDialogToUse(partDetailsDlg);
            if (!subCommands[i].run()) {
                return false;
            }
        }
        return true;
    }
}
static abstract class SubCommand implements Command {
    ParticipantDetailsDialog partDetailsDlg;
    void setDialogToUse(ParticipantDetailsDialog partDetailsDlg) {
        this.partDetailsDlg=partDetailsDlg;
    }
}
static class SetParticipantNameCommand extends SubCommand {
    String partName;
    SetParticipantNameCommand(String partName) {
        this.partName=partName;
    }
    boolean run() {
        partDetailsDlg.setParticipantName(partName);
        return true;
    }
}
}

```

```

        static class ClickOKCommand extends SubCommand {
            boolean run() {
                partDetailsDlg.clickOK();
                return true;
            }
        }
    }
}

class ParticipantDetailsDialog extends Jdialog {
    String partId;
    JTextField partName;
    JTextField partEmail;
    ...
    JButton OK;
    JButton cancel;
    ParticipantDetailsDialog(String partId) {
        this.partId=partId;
    }
    void setParticipantName(String name) {
        partName.setText(name);
    }
    void setParticipantPassword(String password) { ... }
    void setParticipantEmail(String email) { ... }
    void setParticipantAddress(String address) { ... }
    void clickOK() { ... }
    void clickCancel() { ... }
}

```

Note that the run method in ShowParticipantDetailsDialog creates a ParticipantDetailsDialog, but it doesn't call showModal to show it. This is right. If it did, the dialog would be shown on the screen, waiting for a user to input. Then the ShowParticipantDetailsDialog command could not continue to execute its sub-commands.

Other things to test

We have tested whether ParticipantDetailsDialog can handle the case when a new participant is added. We also need to test the case when the participant ID already exists or when the user clicks Cancel. Then, we should test ParticipantIdDialog. Because the behavior of a menu item is too simple. We don't need to test it.

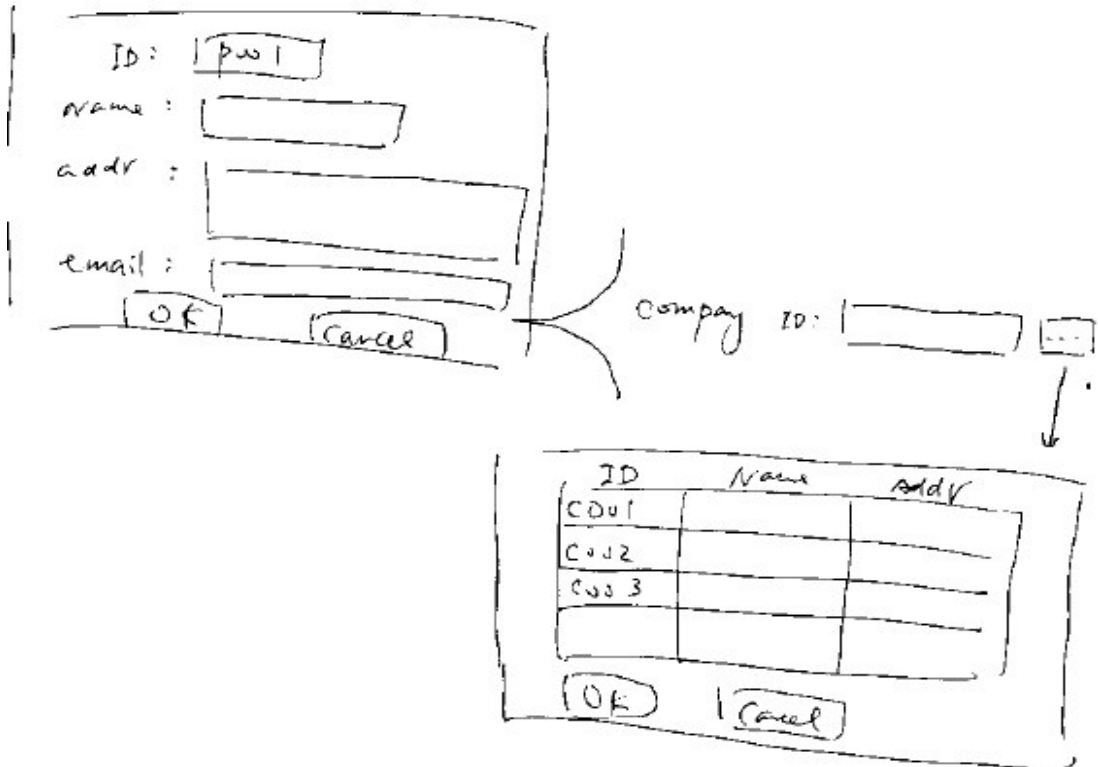
Because the way to test these is the same, we will not talk about it here.

What if one dialog needs to pop up another?

Commonly we need to pop up one dialog from another. For example, ParticipantDetailsDialog may need the users to input the company ID of the participant. To make it convenient for the user, by the side of the company ID edit box there is a "..." button. The user can click this button to display another dialog (let's call it CompaniesDialog) to display all the companies in

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agileskills.org

the system and then chooses one of them, saving the need to input that company ID manually. Below is the sketches you drew on the white board during the discussion:



The question is, how to test this function of selecting a company? Would the test file below work?

```

SystemInit
AddCompany, c001, ...
AddCompany, c002, ...
ShowParticipantDetailsDialogStart, p001
  SetParticipantName, Mike Chan
  SetParticipantPassword, 888888
  SetParticipantAddress, aaabbb
  SetParticipantEmail, mike@excite.com
  ShowCompaniesDialog
  ClickOK
ShowParticipantDetailsDialogEnd
CheckParticipantStored, p001, ...

```

In which, in order to have some companies for a user to select, we use two AddCompany commands to add two companies (with ID of c001 and c002 respectively) to the system. The ShowCompaniesDialog command pops up a CompaniesDialog.

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agileskills.org

However, this ShowCompaniesDialog command doesn't work. In addition to creating a CompaniesDialog, we also need to interact with the CompaniesDialog, e.g., select a particular company and then click OK. Therefore, we change it to:

```
SystemInit
AddCompany,c001,...
AddCompany,c002,...
ShowParticipantDetailsDialogStart,p001
  SetParticipantName, Mike Chan
  SetParticipantPassword, 888888
  SetParticipantAddress, aaabbb
  SetParticipantEmail, mike@excite.com
  ShowCompaniesDialogStart
    SelectRowInGrid, 1
    ClickOK
  ShowCompaniesDialogEnd
  GetCompanyId, c002
  ClickOK
ShowParticipantDetailsDialogEnd
CheckParticipantStored,p001,...
```

In which we use a SelectRowInGrid command to select row 1 in the grid (assuming the first row is row 0), that is, the row representing c002. Then we use a ClickOK command to click OK. In the ParticipantDetailsDialog command we use a GetCompanyId sub-command to check if c002 has been input automatically as the company ID.

This method works. However, we are testing ParticipantDetailsDialog and CompaniesDialog together. As mentioned before, usually it is easier to test each user interface component separately. Therefore, we wish that ParticipantDetailsDialog didn't use another dialog. Does it really have to use CompaniesDialog? In fact, what it really needs is just to get a company ID from someone. Therefore, we can abstract the CompaniesDialog as a string provider or a string source and let ParticipantDetailsDialog use a string source instead of a dialog:

```
interface StringSource {
  String getString();
}
class CompaniesDialog extends Jdialog implements StringSource {
  String getString() {
    showModal();
    return OKClicked() ? getSelectedCompanyId() : null;
  }
  boolean OKClicked() { ... }
  String getSelectedCompanyId() { ... }
}
class ParticipantDetailsDialog extends Jdialog {
  String partId;
  StringSource companyIdSource;
  JTextField partName;
  JTextField partEmail;
  JTextField compId;
  ...
  JButton OK;
  JButton cancel;
  Jbutton chooseCompany;
  ParticipantDetailsDialog(String partId) {
```

```

        this.partId=partId;
        this.companyIdSource=new CompaniesDialog();
    }
    void clickChooseCompany() {
        String companyId=companyIdSource.getString();
        if (companyId!=null) {
            compId.setText(companyId);
        }
    }
}

```

The original test file can be split into two. One file tests ParticipantDetailsDialog:

```

SystemInit
AddCompany,c001,...
AddCompany,c002,...
ShowParticipantDetailsDialogStart,p001
    SetParticipantName,Mike Chan
    SetParticipantPassword,888888
    SetParticipantAddress,aaabbb
    SetParticipantEmail,mike@excite.com
    ClickChooseCompany,c002
    GetCompanyId,c002
    ClickOK
ShowParticipantDetailsDialogEnd
CheckParticipantStored,p001,...

```

The other file tests CompaniesDialog:

```

SystemInit
AddCompany,c001,...
AddCompany,c002,...
ShowCompaniesDialogStart
    SelectRowInGrid,1
    ClickOK
    GetSelectedCompanyId,c002
ShowCompaniesDialogEnd

```

The ClickChooseCompany sub-command can be implemented like this:

```

class ShowParticipantDetailsDialogCommand implements Command {
    ...
    static class ClickChooseCompanyCommand extends SubCommand {
        String companyId;
        ClickChooseCompanyCommand(String companyId) {
            this.companyId=companyId;
        }
        boolean run() {
            partDetailsDlg.companyIdSource=new StringSource() {
                String getString() {
                    return companyId;
                }
            };
        }
    };
    partDetailsDlg.clickChooseCompany();
    return true;
}

```

```
}  
}
```

References

- <http://c2.com/cgi/wiki?GuiTesting>.



Chapter exercises

Problems

1. Write a test file to test whether ParticipantDetailsDialog can handle the case when a user clicks Cancel. Implement the required commands.
2. Write a test file to test whether ParticipantDetailsDialog can handle the case when the participant ID already exists. Implement the required commands.
3. Implement the setParticipantAddress method in ParticipantDetailsDialog.
4. Write some test files to test ParticipantIdDialog. Implement the required commands.

Hints

1. No hint for this one.
2. No hint for this one.
3. There should be a GetParticipantId sub-command that checks the participant ID input by user.

Sample solutions

1. Write a test file to test whether ParticipantDetailsDialog can handle the case when a user clicks Cancel. Implement the required commands.

```
SystemInit
ShowParticipantDetailsDialogStart,p001
  SetParticipantName,Mike Chan
  SetParticipantPassword,888888
  SetParticipantAddress,aaabbb
  SetParticipantEmail,mike@excite.com
ClickCancel
ShowParticipantDetailsDialogEnd
CheckNoParticipantsStored
```

No command needs to be implemented.

2. Write a test file to test whether ParticipantDetailsDialog can handle the case when the participant ID already exists. Implement the required commands.

```
SystemInit
AddParticipant,p001,Mary Lam,123456,abc,mary@hotmail.com
AddParticipant,p004,John Chan,888999,def,john@yahoo.com
ShowParticipantDetailsDialogStart,p001
  GetParticipantName,Mary Lam
  GetParticipantPassword,123456
  GetParticipantAddress,abc
  GetParticipantEmail,mary@hotmail.com
  SetParticipantName,Mike Chan
  SetParticipantPassword,888888
  SetParticipantAddress,aaabbb
  SetParticipantEmail,mike@excite.com
  ClickOK
ShowParticipantDetailsDialogEnd
CheckParticipantStored,p001,888888,Mike Chan,aaabbb,mike@excite.com
```

Implementation of the commands:

```
class AddParticipantCommand implements Command {
    Participant part;
    AddParticipantCommand(Participant part) {
        this.part=part;
    }
    boolean run() {
        ConferenceSystem.getInstance().parts.addParticipant(part);
        return true;
    }
}
class ShowParticipantDetailsDialogCommand implements Command {
    ...
    static class GetParticipantNameCommand extends SubCommand {
        String partName;
        GetParticipantNameCommand(String partName) {
            this.partName=partName;
        }
    }
}
```

```

        boolean run() {
            return partDetailsDlg.getParticipantName.equals(partName);
        }
    }
    static class GetParticipantPasswordCommand extends SubCommand {
        ...
    }
    static class GetParticipantAddressCommand extends SubCommand {
        ...
    }
    static class GetParticipantEmailCommand extends SubCommand {
        ...
    }
}
class ParticipantDetailsDialog extends JDialog {
    ...
    JTextField partName;
    ...
    String getParticipantName() {
        return partName.getText();
    }
}

```

3. Implement the setParticipantAddress method in ParticipantDetailsDialog.

```

class ShowParticipantDetailsDialogCommand implements Command {
    ...
    static class SetParticipantAddressCommand extends SubCommand {
        String partAddress;
        SetParticipantAddressCommand(String partAddress) {
            this.partAddress=partAddress;
        }
        boolean run() {
            partDetailsDlg.setParticipantAddress(partAddress);
            return true;
        }
    }
    static class ClickOKCommand extends SubCommand {
        boolean run() {
            partDetailsDlg.clickOK();
            return true;
        }
    }
}
class ParticipantDetailsDialog extends JDialog {
    ...
    JTextField partAddress;
    ...
    void setParticipantAddress(String address) {
        partAddress.setText(address);
    }
}

```

4. Write some test files to test ParticipantIdDialog. Implement the required commands.

Test if it can return the participant ID input by a user:

```
SystemInit
```

```
ShowParticipantIDDIALOGStart
  SetParticipantID,p001
  ClickOK
  GetParticipantID,p001
ShowParticipantIDDIALOGEnd
```

Test if it will return empty if a user clicks Cancel:

```
SystemInit
ShowParticipantIDDIALOGStart
  SetParticipantID,p001
  ClickCancel
  GetParticipantID,
ShowParticipantIDDIALOGEnd
```

Implementation of the commands:

```
class ShowParticipantIDDIALOGCommand implements Command {
    ...
    static abstract class SubCommand implements Command {
        ...
    }
    boolean run() {
        ParticipantIDDIALOG partIDDlg=new ParticipantIDDIALOG();
        for (int i=0; i<subCommands.length; i++) {
            subCommands[i].setDialogToUse(partIDDlg);
            if (!subCommands[i].run()) {
                return false;
            }
        }
        return true;
    }
}
static class SetParticipantIDCommand extends SubCommand {
    String partID;
    SetParticipantIDCommand(String partID) {
        this.partID=partID;
    }
    boolean run() {
        partIDDlg.setParticipantID(partID);
        return true;
    }
}
static class GetParticipantIDCommand extends SubCommand {
    String partID;
    GetParticipantIDCommand(String partID) {
        this.partID=partID;
    }
    boolean run() {
        return partIDDlg.getParticipantID().equals(partID);
    }
}
static class ClickOKCommand extends SubCommand {
    boolean run() {
        partIDDlg.clickOK();
        return true;
    }
}
static class ClickCancelCommand extends SubCommand {
```

```
        boolean run() {
            partIDDlg.clickCancel();
            return true;
        }
    }
}
class ParticipantIDDIALOG extends JDialog {
    ...
    JTextField partID;
    ...
    void setParticipantID(String id) {
        partID.setText(id);
    }
    void clickOK() { ... }
    void clickCancel() { ... }
}
```