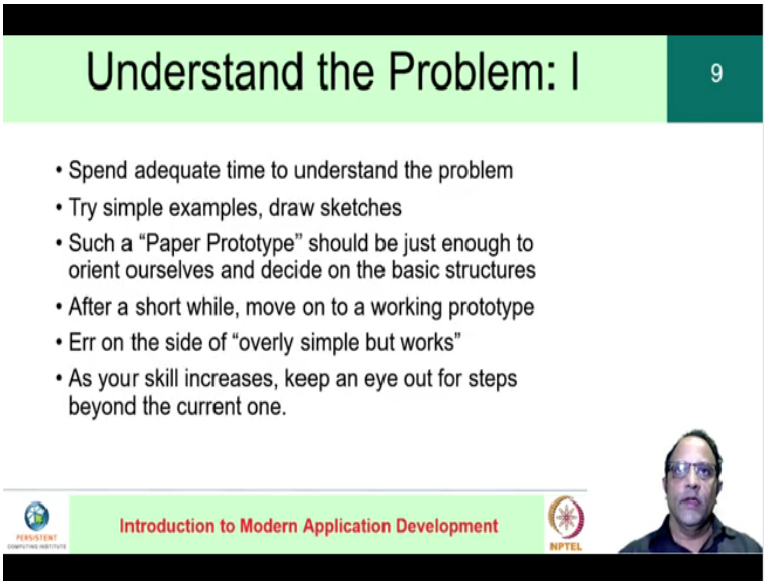
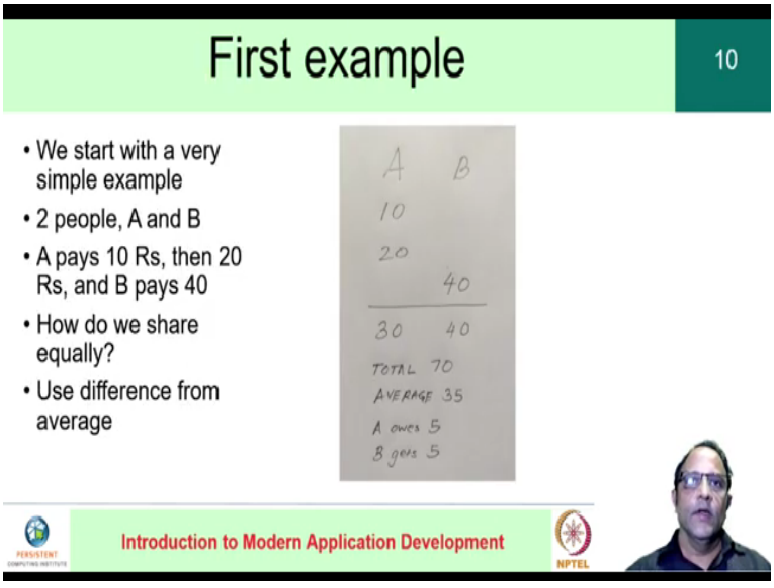
Introduction to Modern Application Development

Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

Lecture-03
Introduction To Modern Application Development Part 3

(Refer Slide Time: 00:15)





So, here is our first example continued from the last lecture. We start by looking at a case where there are only 2 people *A* and *B.* And the scenario will discuss is as follows:

1. *A* pays 10 rupees first for some joint event that *A* and *B* participate in.
2. *A* pays 20 rupees for another activity.
3. *B* pays for 40 rupees for some joint activity.

And then after this sequence of activities, we want to make sure that they have shared money equally.

Clearly, this is a very simple case indeed, just to make get a basic idea of what it is that our application should do. So, let us see what happens. How do we figure out the way to share the expenses equally in this case? The explanation is obvious and as follows:

1. Take the total of the expenses of both.
2. Find out what the average is.
3. Since everybody should pay the average, subtract the paid amount from the total paid by each of them to get to know who owes whom and how much.

In this example, since everybody should pay the average, *A* has underpaid and *B* has overpaid and therefore, *A* owes *B* 5 rupees, and in that case everything is balanced.

So, this is of course, an excessively simple case, but merely the analysis of this example is not the what we are concerned about; the point of this discussion to make it very clear and make it absolutely sure how the simple cases work and build up the algorithm for the program from there while checking every step along the way.

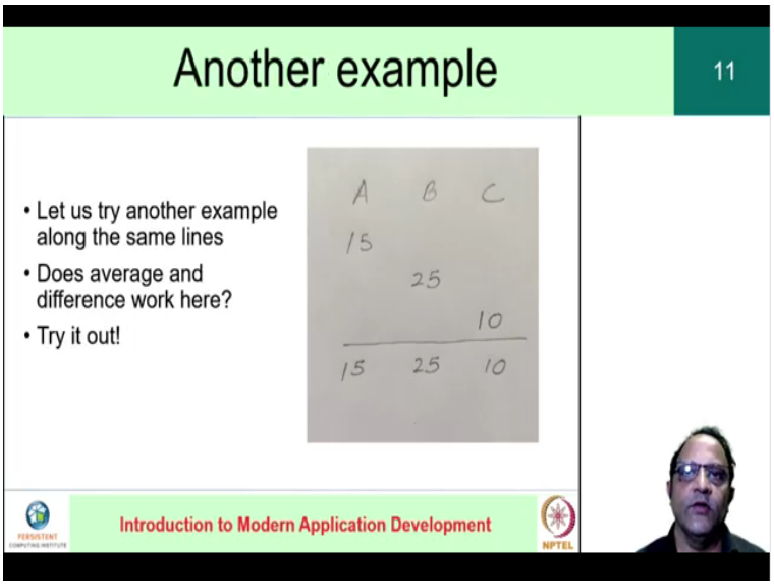
So, now, what we do is that, even for a very simple case like this, we lay out these figures explicitly. In our little two people example and for the basic scenario we are considering, we might think it does not matter much. This is true, however our point here is not to talk about

simple arithmetic, but to show you how to organize ideas in a way that can scale up to far more complicated and complex situations.

This style of thinking is called back-of-the-envelope calculation. It is often useful in clarifying various situations. The oversimplification that we are doing is a deliberately chosen strategy to make sure that we make progress step by step.

If you are interested in this kind of thinking, there is a fun book called *Street Fighting Mathematics* by Sanjoy Mahajan, which shows how to tackle far more complicated situations in this style. This kind of thinking, when used for complicated situations is also known as the Fermi principle, named after a physicist who was famous for doing simple calculations to figure things out. Of course, in this case, none of that is needed. Still, we will make sure that we progress step by step.

(Refer Slide Time: 03:01)



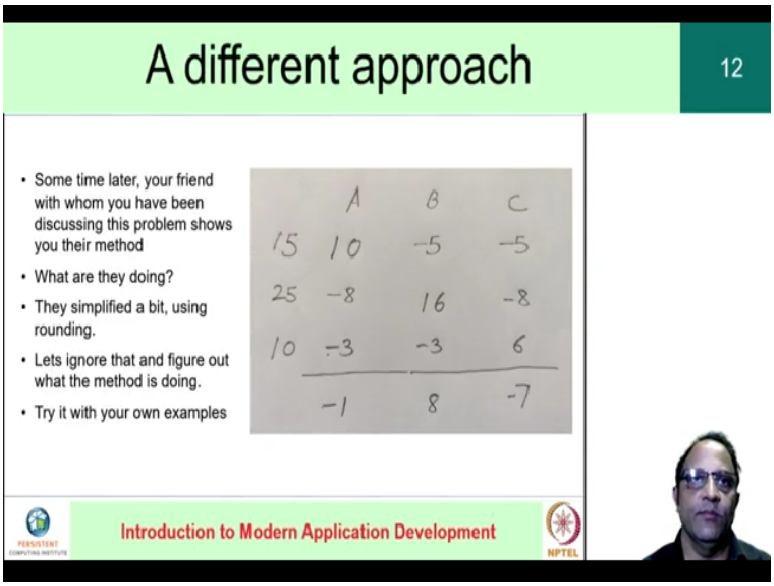
Okay, that commentary aside, let us get back to our main program. The next step we do after our 2 people example is to see whether the same strategy is liable to work with 3 people. So now we have 3 people, *A B* and *C,* and we have 3 activities. *A* pays 15 rupees, then *B* pays 15 rupees, that *C* pays 10 rupees. And at the end of this sequence of activities, we will try to divide the money

equally. The first thing we try is to use the same averaging method that we use before. Do you think it would work with this example? You can try it out to see what happens in this slightly more complex setting before proceeding further.

We can also ask ourselves if it will work for even more complex settings such as multiple people paying? So, we just said that we will start off with simple examples, then why am I raising the question of a more complex setting. Should we even be thinking about this right now? This is a very useful question to ask. We have agreed that we would not consider the complicated settings, but things are not quite that simple.

We need to carry at the back of our mind that there will be some choices of features that we will be making in the future, and it is useful to have them at least on the periphery of our vision, so that you can make sure that the choices we make are *not oversimplified*. In general, though, you should err on the side of *"overly simple but works"*.

(Refer Slide Time: 04:20)



And as your skill develops, you can keep an eye out for the next steps. For example, suppose you are discussing this problem with a buddy of yours. And he goes around and thinks about it and comes back and shows you their method. That is, he shows you a sheet that looks like this *[as*

shown in the slide at time 4:20], there are *A, B,* and *C*, and somebody has paid 15, somebody has paid 25 and somebody has paid 10 and now they are doing some sort of splitting along every row. And somehow when they complete the addition of everything, and the answer seems to magically pop out, that *A* owes some money, *B* should receive some money and *C* also owes some money.
So, can you figure out from this diagram, what is it the idea that your buddy has come up with? Of course, they have simplified it a bit, and they are using rounding. So, the answers are not really exact. But let us ignore that for now and figure out what it is the employed method here.

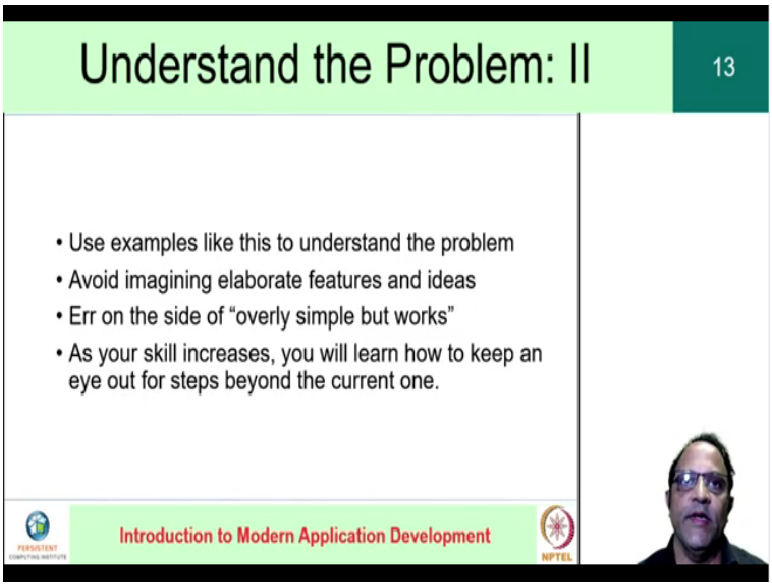
Think about these methods and try them out with your own examples. These are simple examples of how we go around understanding problems. The strategy can be understood in three simple points:

1. You try out examples.
2. You sketch them.
3. You see what happens.

And keep the thought around to understand that you have chosen a reasonably decent model of whatever it is that you have set out to do.

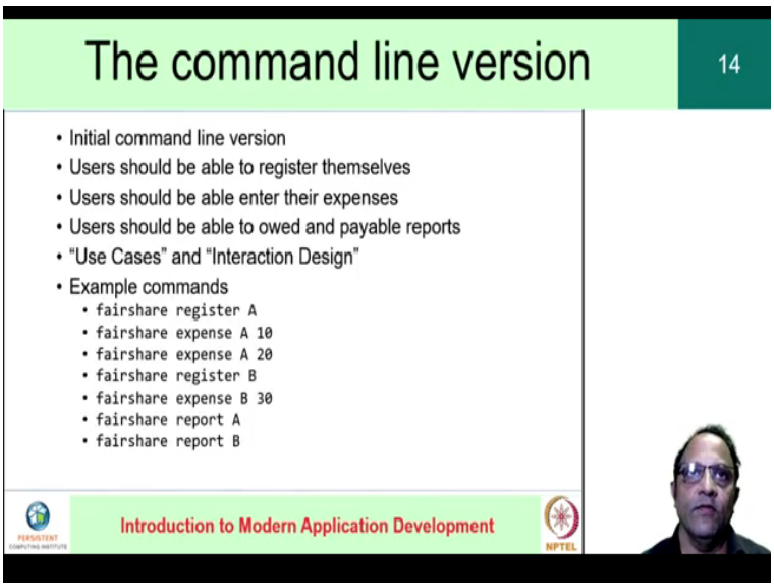
Okay, at this point, we have some examples, and we have some idea of what it is like to actually implement the logic of our program. Now, let us look at one other aspect of the system, which is to see what it should feel like to use the app that we have designed.

(Refer Slide Time: 06:04)



Here, again, we are going to use that approach that we talked about having understood the problem, and having taken the strategic decision to start simply, what we will do is to build a command line version first.

(Refer Slide Time: 06:11)



This will also help us clarify any issues, and it also fits in with the amount of experience we have. We do not yet know how to design a modern app, but we do know how to implement a command line app. And that is what we are going to use.

So, let us start. Our command line program will be used by users who first register themselves and tell the system their name, then they will tell their expenses. And after they are done with their sequence of expenses, they can ask the system for a report and then they should get some details of how much they owe. Now, in a complicated setting, this kind of thinking that there are 3 steps to take, there is first registration, there is expensing and then there is reporting, these must be the 3 steps that our app should take. And this kind of thinking you can actually make far more systematic.

You may have heard of terminology like *use cases* and *interaction design* and so forth. The idea behind these more systematic methods of thinking about applications is as follows. In the case of *use case analysis*, you figure out in an organized way, how is it that your program will be used, because ultimately, as a programmer, you have to learn to think from the point of view of the user, not from the point of view of the implementer alone.

So, how the program will be used will affect the structure of the program and that is one of the important things we need to know it. The other important reason is that people are going to live with this program for a while, and they must find it easy enough to use, and it should be somehow natural to whatever the activity there is; and this part can be thought about under the heading of *interaction design*.

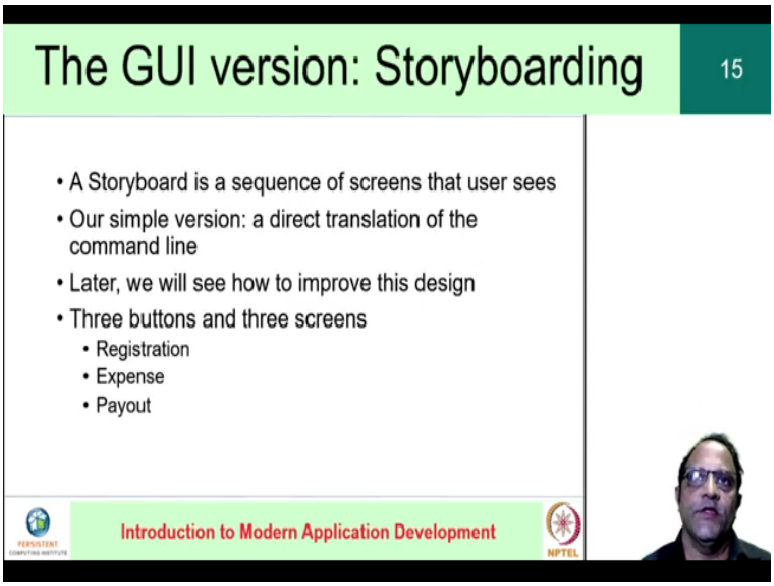
Lastly, the third aspect is how are the looks and what is the feel like. So, this is taken care of under the title user interface design, which is usually discussed in the context of GUI as graphical user interfaces, where we think about how we present the app to the user. All these 3 aspects, *the use cases, interaction design,* and *graphical user interfaces* are interdependent. There is much more to talk about these things, but for now, we will go ahead with our basic app.

So, as you can see, if we imagine what the command line app is like, for the simple examples that we have discussed, the usage of the app will look as described in this slide.

- Users first call up the FairShare program.
- Then A registers themselves.
- Then A enters their expenses.
- And then B registers themselves.
- And then they add their expense.
- And finally, user asks for the report at which point the app should print out who owes how much to whom.

So, this is a sketch of what it is like to use our app. One reason we begin here is because this is what we know. But that is not really the end of the story! This is not the actual app we want to build for our users. Therefore, we will give some thought to what the GUI version of the app should look like.

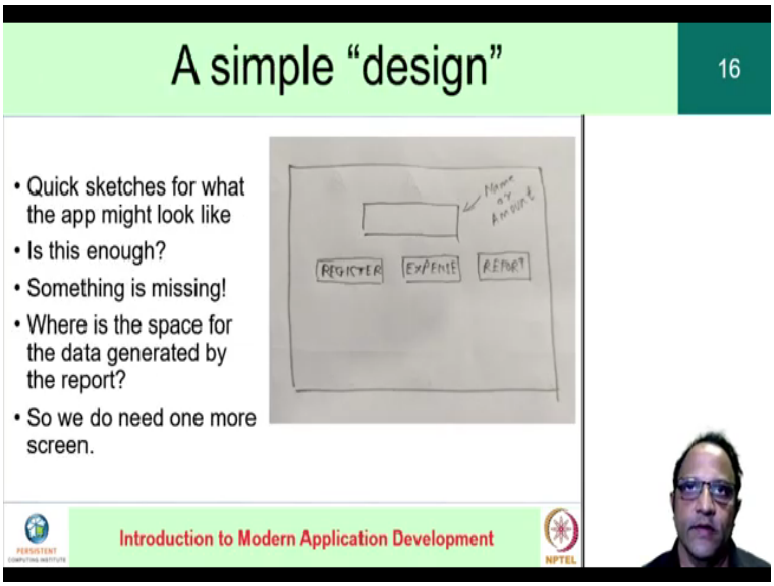
(Refer Slide Time: 09:38)



For the GUI version of the app, we imagine the app in the form of the screens that it displays. The sequence of screens that app can display is called a *storyboard*. Our app will have 3 screens, but the screens will look fairly similar. In our first design, we will just straight away copy whatever it is that we do for the command line program, after some experience with the app, we can figure out how to improve the interactions that are experienced by the user, and redesign our screens accordingly.

With this in mind, our design is straightforward: there are 3 buttons in the app, just as there are 3 commands. And when a button is pressed, you go to the next screen, and the next screen does not necessarily need to look different. Although conceptually, it is a different screen. We might imagine that for the case that we are thinking of, there is not much complexity to implementing it, to get a better idea of what it is that we set out to do.

(Refer Slide Time: 10:46)



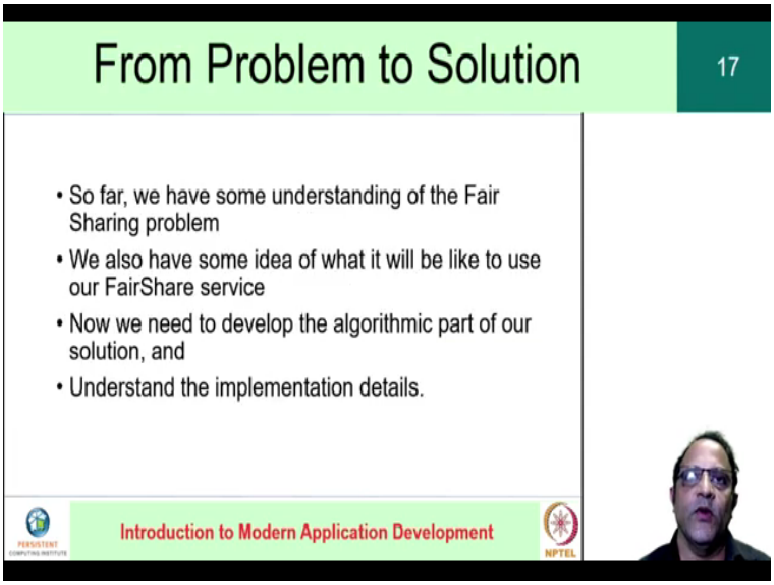
Again, let us indulge in some paper prototyping, and accordingly we will do a sketch of our app. So, here is a quick sketch for what the app might look like. There is a **form** where you can enter the **name of the person** or the **amount**. There are 3 buttons for the following functionalities:

- *Registering a person*
- *Adding the expense*
- *Asking for the report*

So, think about it - this looks reasonable. But is this really enough? Are we missing somethings? Well, Yes! For instance, where is the space for data generated by the report? It is not on this screen, which means there must be another screen, where this kind of data can be shown! There is another thing which is something we have to do, there is a distinction between registering a name and then asking for an expense – hence we should have two different screens for both the user activities. We have also said that the registration of the user uses a name and the expense will ask for a name, therefore, we have to do something in case there is not a match.

So just thinking about it in terms of screens like this, and drawing simple diagrams will show us at least a few things that we need to do, that the paper prototyping has revealed to us. And this is where we can get started taking the next steps to the design of the app. So, just simple things like this and giving some adequate amount of time to understand what it is we have to program, and what it is like to use such an app will go a long way towards helping us make a better and useful design.

(Refer Slide Time: 12:25)



So, at this point, let us summarize what we have done: we have acquired some understanding of the fair sharing problem, we were acquired some understanding of what it is like to use our service, both on the command line and on the screen.

By the way, you might think of the command line version as a sort of a thing that you have to do because that is the only thing you know how to do so far. But that is not the sole reason why you use the command line. Command line versions of graphic apps can be very useful in testing the algorithmic parts of our solutions as well.

Okay, so with that summation, we will move on to develop the algorithmic part of our program and move on to understand the implementation details. This we will do in the next part of our session.