

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-02
Introduction To Modern Application Development Part 2

(Refer Slide Time: 00:11)



Where do App ideas come from? 6

- Many apps that we find useful are computerized versions of activities people already do
- Example 1: Restaurant food delivery once required a phone directory, and was limited to a small area. With Swiggy and Zomato, almost as simple as being at the restaurant
- Example 2: Taxi companies and booking over phone. With Ola and Uber, it is like hailing a taxi on the road
- Creators of these apps *observed a common need* and saw how to do it more simply with new technologies.

Introduction to Modern Application Development NPTEL

Prof. Aamod Sane

Now, that we have figured out the structure of the course, let us try to understand where it is that people get ideas for their apps. Some types of apps like games are invented by people who appear to have a certain viewpoint or style of thinking. These are people like cartoonists, and writers and so on, who have a way of thinking about the world that is not so easy for others to grasp.

But many of the apps that we find useful come about from observing what people are already doing, and finding out ways to do that activity better by using computers. Examples of these kinds of apps are the things that we use in daily life. For instance, there are food delivery apps

like Swiggy and Zomato, taxi apps like Ola and Uber, or map and driving direction apps like Google Maps, or OpenStreetMap.

Once upon a time, there was something called a telephone directory. This was a big book that was delivered to you by the phone company, where you would look up the name of a restaurant, call up the restaurant, and hope that they would deliver food to you. Of course, this was limited to the restaurants that were near enough to your home. Today, the Swiggy or Zomato app with their delivery service makes it possible for us to order food from relatively distant places.

And it gives us real time updates of various stages of our order, as well. It is also fun to look at all the stages that happen while you are waiting for the food at home. Similarly, once upon a time, you had to know about taxi service companies, you have to know their phone numbers. You would have to call them up often a day or so in advance, and then book a taxi. Today, with taxi hailing apps like Ola and Uber, drivers and riders work almost directly with each other.

And the middleman, here the app company, is almost invisible to either party, except when some issue arises. The creators of these apps observed a common need, saw how to implement it using apps and then build a service around this idea. Initially, these ideas were inspired by just plain copying or mimicry of existing services that were based on older technology. But as these apps and the companies building them understand the domain better, and understand better how to use today's technology, the app-based alternative goes far beyond what was possible in the older service.

The app that we will build in this course is based on the similar idea, we will observe a common need and we will set out to solve it and build an app that we can then make available to people as a service. So, what is this app we are dealing with.

(Refer Slide Time: 03:12)



Our App: FairShare 7

- A service to help roommates fairly share their expenses for shared activities
- Simple scenario to begin with
 - Roommates share food, water, cable tv, trips to restaurants and other expenses
 - Assume that for every joint expense, one person pays for the entire activity at any one time
 - At the end of the month, they get a report for money owed and money receivable.
- Consider more complicated scenarios later
 - Several people different amounts for joint activity, instead of one person
 - Only a few roommates participate in some activity, so only they should pay
 - Some person defers their payment to next month
- In any software design situation, start simple and evolve!

Introduction to Modern Application Development

NPTEL

The app that we have designed is called FairShare. FairShare is an app that is a service to help roommates to decide how to split up their expenses, and everyone ends up paying a fair share of all the bills that they have shared jointly. We will start with a service that solves for a very simple situation. The assumptions that we make are like this, roommate share expenses on food, water, cable TV, trips to the restaurant and so on all the activities that they might do jointly.

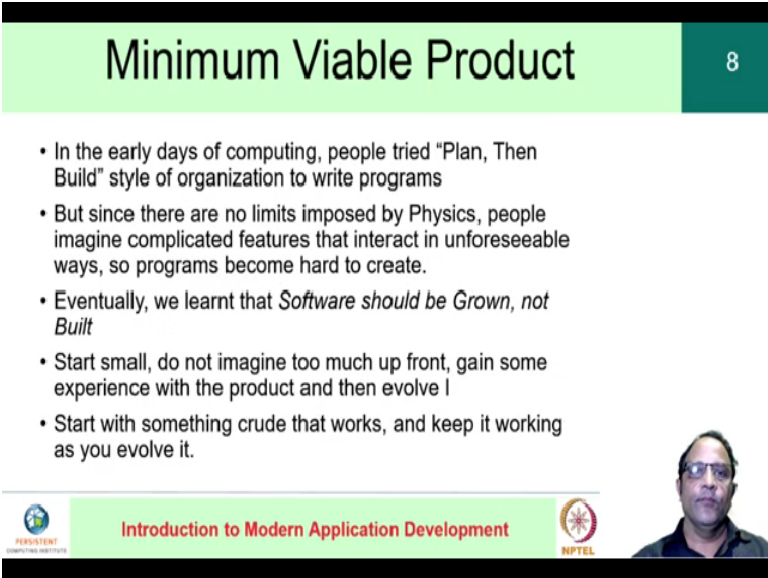
We will assume that for every such joint activity, one person pays for the entire activity at a time. At the end of every month each roommate can get a report that tells them who owes money and who should receive money. So that everyone ends up paying the same amount of money. As you might imagine, in time we can add more complex features to this service. For example, several people might chip in different amounts of money to a single activity, instead of our assumption where only one person pays everything.

Another case is that a few roommates participate in some activity instead of every single roommate participating. In that case, equal sharing applies only to those roommates, while the non participants do not have to pay anything as far as that activity is concerned. This is not a case that our simplified scenario deals with. A third possibility is that some person defers their

payment at the end of the month, and will agree to pay some time later. Our app could be adjusted to track this situation as well.

There are many other such ways you can imagine in which we can make the app service far more complicated. However, in any software design situation, it is best to start small and evolve the program in time. This is why we have laid out the discussion of our app in the manner we have. It is the simple features that are easy to understand that we will begin with first. This idea of gradually growing the program instead of building it was discovered with some effort.

(Refer Slide Time: 05:36)




Minimum Viable Product 8

- In the early days of computing, people tried "Plan, Then Build" style of organization to write programs
- But since there are no limits imposed by Physics, people imagine complicated features that interact in unforeseeable ways, so programs become hard to create.
- Eventually, we learnt that *Software should be Grown, not Built*
- Start small, do not imagine too much up front, gain some experience with the product and then evolve it
- Start with something crude that works, and keep it working as you evolve it.

Introduction to Modern Application Development

PERSISTENT
UNIVERSITY OF TECHNOLOGY

NPTL



In the early days of computing, people tried to build programs following the style used in other engineering fields. They would attempt to decide what to build, create plans and then implement the program. The analogy they were following was with fields like civil engineering, where one draws up plans, create blueprints and then hire subcontractors to work on various pieces, but we discovered, by “we” here, I am in the entire field really, that it is much harder in software to decide what to build and how to build it.

The reason seems to be that there is no nature or physics to impose any limits on what can and cannot be done. The result is that people end up imagining all sorts of features. And it is not just the features themselves, but the way these features interact with each other, that ends up becoming immensely complicated. This is much more so than what can happen under the limits that nature creates. And therefore, fields which rely on nature to put boundaries on, in some ways, have an easier time of deciding what to do because of the very limitations that exist in what can be done.

Eventually we learn that software should not be built, it should be grown. A well-known book called *Mythical Man-Month* by Fred Brooks talks about this discovery. As time went by, we found better ways to organize ourselves beyond what was discussed in *Mythical Man-Month*. The methods that were discovered have names like XP, Scrum, Kanban, which you may have probably heard of.

But whatever their details, there are things that these methods have in common, and perhaps the most important is the single lesson that says: **start small**. Following points are worth remembering:

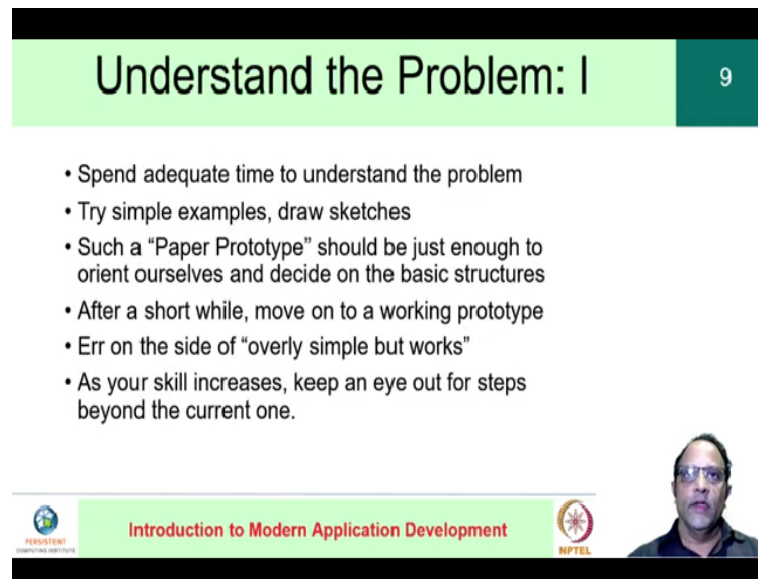
- Do not imagine too much upfront.
- Get some experience with the product.
- Then evolve it one step at a time.

But short name for this idea is **minimum viable product**. That word MVP is a play on the acronym for most valuable player awards in cricket and other sports.

So, we will use the same style as we learned in this lesson. We will start with a very simple scenario, which is crude but it works, we then add more features that seem necessary after some experience. In each step, however, our software will always work. It should never be the case that we will have a long period during the development such that our program works only at the end of that duration.

This fits in well with our overall approach, which is to start with a command line program and end up with a modern app. So that is the general idea. The next step, having specified what it is that we plan to do, and the idea that we will start by building a minimum small app that we can understand. Let us take the next step.

(Refer Slide Time: 08:39)



The slide is titled "Understand the Problem: I" and is numbered 9. It contains a list of six bullet points: "Spend adequate time to understand the problem", "Try simple examples, draw sketches", "Such a 'Paper Prototype' should be just enough to orient ourselves and decide on the basic structures", "After a short while, move on to a working prototype", "Err on the side of 'overly simple but works'", and "As your skill increases, keep an eye out for steps beyond the current one." At the bottom of the slide, there is a video inset of a man speaking, and a footer bar with the text "Introduction to Modern Application Development" and logos for NPTEL and Persistent Systems.

- Spend adequate time to understand the problem
- Try simple examples, draw sketches
- Such a "Paper Prototype" should be just enough to orient ourselves and decide on the basic structures
- After a short while, move on to a working prototype
- Err on the side of "overly simple but works"
- As your skill increases, keep an eye out for steps beyond the current one.

So, the next step is to understand the problem in detail. The first rule is to spend adequate amounts of time to understand the problem. What we do is we try out some of the scenarios by hand and see what the details really are like. A fancy name for what we are doing would be “paper prototyping”.¹ This is just another way of saying that use paper to think the ideas through; our paper prototype will be just enough that we can orient ourselves about where we are in trying these ideas out. It will help us invent the basic algorithm, and in a short while we will also be able move on to the implementation.

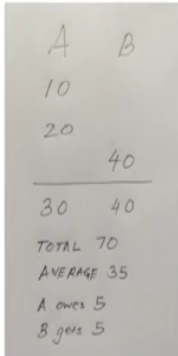
¹ **Paper prototyping** is a process that helps developers to create software that meets the user's expectations and needs by creating rough hand-drawn interfaces and designs to prototype followed by testing.

(Refer Slide Time: 09:24)


First example



10

- We start with a very simple example
- 2 people, A and B
- A pays 10 Rs, then 20 Rs, and B pays 40
- How do we share equally?
- Use difference from average



A	B
10	
20	
30	40
TOTAL 70	
AVERAGE 35	
A owes 5	
B gets 5	



Introduction to Modern Application Development

So, to do this, let us come up with some examples. Here is our first very simple example, there are only 2 people *A* and *B*. And the scenario we will understand 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 event.
3. *B* pays for 40 rupees for some event that again both of them participated.

So, at the end of this sequence of events, *A* has spent 30 rupees *B* has spent 40 rupees, and the question is how do we share equally. Notice that we have taken an extremely simple example to make sure that we will evolve the basic idea and it will work for absolutely basic situations.