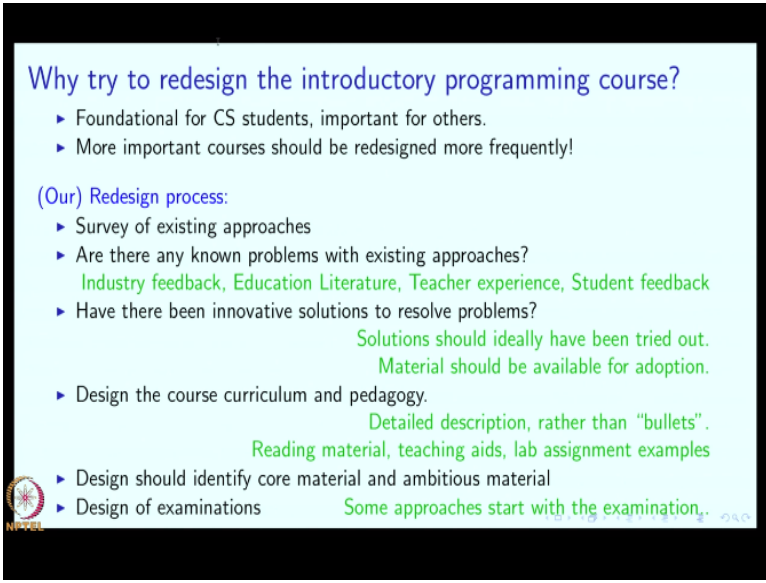


Design and Pedagogy of The Introductory Programming Course
Prof. Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture – 22
Summing Up

Hello and welcome to the last lecture in the course on design and pedagogy of the introductory programming course. Today, we are going to quickly go over, quickly review what we have done and conclude.

(Refer Slide Time: 00:40)



Why try to redesign the introductory programming course?

- ▶ Foundational for CS students, important for others.
- ▶ More important courses should be redesigned more frequently!

(Our) Redesign process:

- ▶ Survey of existing approaches
- ▶ Are there any known problems with existing approaches?
Industry feedback, Education Literature, Teacher experience, Student feedback
- ▶ Have there been innovative solutions to resolve problems?
Solutions should ideally have been tried out.
Material should be available for adoption.
- ▶ Design the course curriculum and pedagogy.
Detailed description, rather than "bullets".
Reading material, teaching aids, lab assignment examples
- ▶ Design should identify core material and ambitious material
- ▶ Design of examinations
Some approaches start with the examination.

So we could ask I guess why try to redesign the introductory programming course and we said that it is foundational for CS students and it is also important for others and since it is an important course, it should be redesigned frequently. The more important courses should be redesigned more frequently. Here was our redesign process in brief. We began by surveying existing approaches.

And then we are asked are there any known problems which have been identified in the existing approaches and for this we used various mechanisms or various sources. So we looked at the industry feedback, well industry feedback and also a feedback by bodies which survey students and then we looked at the educational literature. This is a very, very useful, very, very reliable source of information.

We looked at teacher experience, so in the present case this includes my own experience as a teacher, also my experience and information I get by talking to other teachers and finally design shaped by the feedback that I have received from students. So this feedback is formal feedback like the feedback which gets collected at the end of the semester and also in total feedback that is feedback I have received in talking to students.

So we started our entire exercise by looking at previous work and by where there are any known problems and then we also asked what can be the solutions to these problems. So an important idea over here is that we should not really go with paper solutions okay. Theoretical solutions are not adequate. We should really look for solutions which have also been tried out.

So in fact in our exercise we did this and will come to the details in a minute. The solution should be tried out and if it needs any material or whatever it certainly will need documentation, so that documentation should be available. So that others can actually follow and do what is being prescribed.

So having looked at this preliminary part which is survey, the literature survey, how the course has been taught, figure out what the difficulties are, figure out what the possible solutions to those difficulties are. Then, we design the actual course curriculum and pedagogy in light of what we have known in terms of the work that has been done earlier and here when we do the design of the course, we should really provide detailed description rather than just bullets.

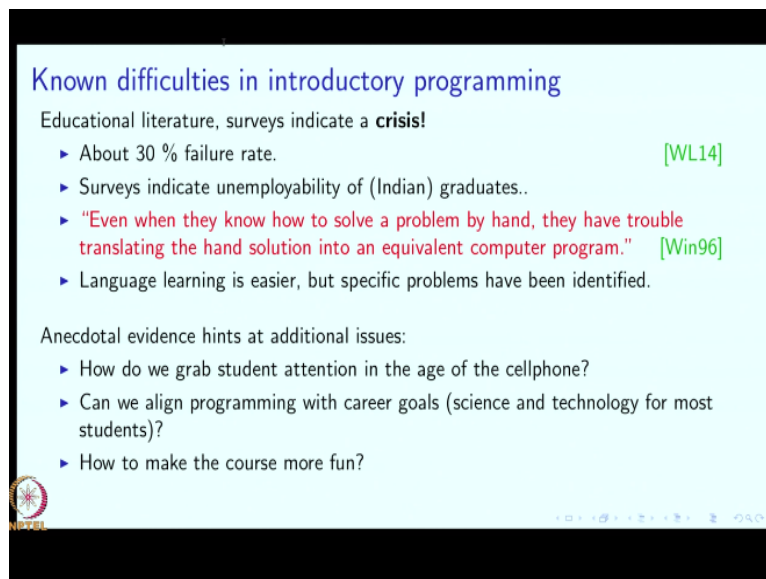
So the same bullet point can be interpreted in many ways by different users and so we really should have a more specific interpretation and to get that interpretation across we need to provide a detailed description. This detailed description can take the form of reading materials, teaching aids, examples of lab assignments. Then, the design should identify core material and ambitious material.

This is because different universities will have different constraints depending upon the size of their semester, the kinds of students they get. So if we have to be useful to all of these people then we should say that look this is something that has to be taught, this is something that we should teach if you have more time and we should design examinations as well. In

fact, there are some approaches to the design process which say that begin from the examination.

So if this is going to be your examination then how do you design the curriculum around it? We did not take that route we instead ask the question look what is that we want to teach and what are the difficulties in that but in any case we must provide something by way of examinations, some suggestions, some discussion of possible problems and that is really important.

(Refer Slide Time: 05:23)



Known difficulties in introductory programming

Educational literature, surveys indicate a **crisis!**

- ▶ About 30 % failure rate. [WL14]
- ▶ Surveys indicate unemployability of (Indian) graduates..
- ▶ "Even when they know how to solve a problem by hand, they have trouble translating the hand solution into an equivalent computer program." [Win96]
- ▶ Language learning is easier, but specific problems have been identified.

Anecdotal evidence hints at additional issues:

- ▶ How do we grab student attention in the age of the cellphone?
- ▶ Can we align programming with career goals (science and technology for most students)?
- ▶ How to make the course more fun?

So let us begin with what we said regarding the known difficulties in introductory programming. The educational literatures and the surveys of introductory programming indicate a crisis. This crisis is manifested in terms of a 30% failure rate. So there are several papers, this is one paper I have cited and then the surveys by independent agencies indicate unemployability of graduates and in these surveys I am referring to Indian graduates.

So aspiringminds.com is one such survey. Then, here is sort of the key difficulty. This is articulated by Winslow. So he says and I think most people would agree with this that even when they our students know how to solve a problem by hand, they have trouble translating the hand solution into an equivalent computer program. So they can do things manually but when it comes to writing programs they have problems.

So this I think is the heart of the problem and it has been articulated and it has been acknowledged as the key problem in many, many articles in the educational literature. The

educational literature also says that language learning is easier than this but even in that there are some very specific problems and we have gone over some of these problems during the course.

Anecdotal evidence hints at some additional issues. For example, the most obvious I guess is that students in the age of cell phone are busy with lots of things. They have many, many distractions and they have many, many distractions which relate to cell phones which actually contain a computer in them and which contains exciting graphics and exciting media. How do we grab their attention and get them interested in programming?

Another age old problem I guess that we must align our programming education with the career goals. So the career goals for most of the students that we are talking about have to do with science and engineering and so somehow our programming course should relate to that and last but not the least we must make our courses more fun. If they are not fun, we will lose several students.

If they are fun, then we will see a lot more enthusiasm but of course fun cannot be the central aspect and fun does not have to be frivolous. Fun can be sort of very solid, fun can be genuine excitement, fun can be a sense of accomplishment, so those are all the aspects. Finally, this is sort of a peculiar characteristic of introductory programming that the initial weeks tend to require some amount of delivery of information.

And in this delivery of information, active learning is often hard to implement and so if we could do that then we could really get students to learn and get students excited. So we identified some difficult issues and then we resolved them and let me summarize what we said in this regard.

(Refer Slide Time: 09:20)

Difficult issues and their resolution

"Students cannot design programs even when they can solve problems manually"

Resolution:

- ▶ Students knowledge of algorithms is intuitive/geometric, while what is needed is algebraic/formal.
- ▶ Suggestion: we help students make their understanding more explicit and translate it into programs.

"Need to attract student attention (quickly), align to career"

Resolution:

- ▶ Scaffolding: Graphics library and repeat statement
- ▶ Incorporation of programming problems from diverse areas: science and technology, commerce, art, fun..

NETEL

So the first and the most important difficulty issue was students cannot design programs even when they can solve problems, the same problem they can solve manually. How did we resolve this? So we said that student knowledge of algorithms is intuitive and geometric while what is needed is algebraic and formal and we suggested that we help students make their understanding more explicit, ask them to introspect.

And then we also help them in translating that more precise knowledge into a program. So this we identified as a very specific step that we needed to take. Then, there was the issue of we need to attract student attention and we need to do it quickly and we need to align it to the career. We need to have students attentive to us from day 1. So we resolve this by saying that we are going to use scaffolding which is an idea in this literature.

And the specific scaffolding we suggested was the use of a graphics library and a repeat statement. Then, we also suggested that we should incorporate programming problems from diverse areas. Science and technology, commerce, art, fun, even fun is important. We can have problems in which students will draw interesting designs but if the designs have structure then students will be learning a lot.

If the designs have periodicity, then the students will be learning how to use iteration. If the designs have self-similar patterns, then the students will be learning how to use recursion. So things can be fun yet things can be very, very educative. So then we came to the course design.


(Refer Slide Time: 11:19)

Course design - 1

At a high level all course designs look similar.. Differences are in the emphasis..

Some salient features of our design

- ▶ Key skill to teach: translating manual algorithms to computer programs.
Introspecting over "what you know", make plan/invariant
Learn to reason about values computed, variables needed..
- ▶ Explicit teaching of "Common sense" based algorithm design techniques but no advanced techniques.
"try all possibilities", "save work in similar computations"
- ▶ Motivate everything in the language of the student.
If hard to motivate, don't teach!
- ▶ Encourage use of simple language constructs.
- ▶ Explain semantics by telling what actually happens rather than by notional explanations.
- ▶ Use of graphics, repeat.



And I guess I should say here that at some high level all course designs look similar but the (()) (11:25) is in the details as the say, so if we have to look at the details and then there we know what is being emphasized and what is not emphasized and then you see how the course designs can be different and then you can really understand what we are actually suggesting.

So here are some salient features of our design. The first is that in our opinion, the key skill to teach is translating manual algorithms into computer programs. This has two parts, first is just getting the student to introspect. So we said that the student already knows a manual algorithm and yet cannot program. So let us get the student to introspect, let us get the student to figure out what exactly he or she does when solving a problem.

And then we should teach how to take that specification and convert it into a computer program. So introspection is important and then learning to reason about the values which are being computed will tell you whether you need loops, what kind of variables you are needed and we said that this is an important enough set of topics that we need to pay special attention to it and we need to dedicate lecture hours to this.

We also said that explicit teaching of common sense based algorithm design techniques should be done but we should not really teach any advanced techniques. So the common sense based algorithm design techniques which we recommended were something like try all possibilities or save work if you are doing similar computations. So if you have done one computation already and you are doing something similar then see if you can do the common parts to get there something like that.

Then, very important and this is very important in general but we want to really, really focus on this which is that we have to motivate everything in the language of the student. We cannot say that look experts think this is useful. We have to say, we have to persuade the students that what they are teaching must be important and this is difficult because students do not know the world in some sense.

They do not know the world of computing. They may not have the imagination to figure out what is going to be important but we believe it can be done and we have given examples of it. So we go through the process of educating students about how computers are used and in parallel build up the motivation as well. So we are really very emphatic about this to the extent that will say that if you cannot motivate something then perhaps you should not even bother to teach it.

Because even if you teach it, if the students do not know why they are learning it they will just forget it. They will go through the process and that material will not stick. So motivation is really important and motivation has to be given at every stage, every language construct must be motivated okay, every problem that you ask them to do will be done better if the students see that the problem is not an artificial problem.

But it is an interesting real life fun problem. As far the language was concerned we said that languages can be very complicated but we do not have to teach all the complexities. We said that it is almost like spoken language where simplicity of how you speak is important. So even here the same idea goes and we said we emphasized simplicity. Now as far as explaining the semantics is concerned, we said that you can give a notional explanation in terms for human analogies.

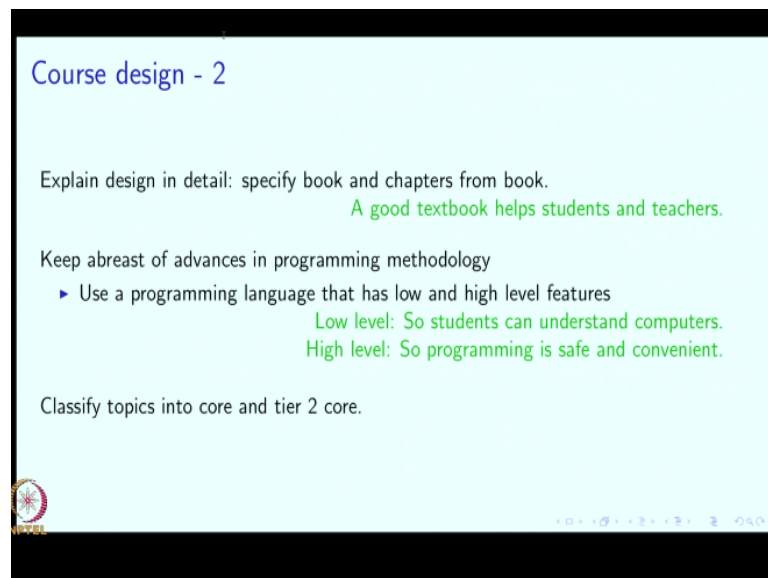
So you could say that a computer is like a human being but our recommendation was that rather than saying this we should tell the students something which is very close to what happens, something in principle that might be happening. Of course, modern computers are extremely complicated. So we cannot tell them exactly what happens but we still can give the gist of what is going on.

So there are several examples of this which we gave for example we could say that a variable is actually a piece of your memory okay. It is a very specific set of capacitors if you will in your memory or we said that when a function call is made then an activation frame is created. So our rationale here was that students can understand these explanations. They are willing to understand these explanations.

Because they want to know, they get a kick out of learning what actually is going on and furthermore if we teach them real explanations then they are unlikely to make misunderstandings. So if we give them human explanations, then they will say oh really what they will say boils down to somehow or the other computers must have common sense. So we do not want such misconceptions and therefore we recommend making as real explanations as possible.

Then, a very important feature of our design was use of graphics and repeat and we have discussed how these things help us in explaining concepts, how these things help us in getting started very rapidly and keeping the student attention. Then, we explained our design in great detail and we believe that anybody who is producing a course design should give a detailed design.

(Refer Slide Time: 17:58)



So they should specify a book if possible and chapters from the book that is extend of detail which we feel is important and we did do that. So we have given a textbook but if you are going to do a design, if you do not choose this textbook, the recommendation here certainly is that find a textbook and we have said how do you go about finding a textbook? So in addition

to that as far as course design is concerned, the textbook that you recommend should be off your style.

In the sense that it should emphasize it should indicate what kind of emphasis you need. So a text book is helping the designers as well as the teachers and students. Then, we said that programming has evolved over time and we should be picking a modern programming language but we said that it must have two characters okay. So it must have high level features as well as low level features.

It should have low level features because low level features can tell you what is going on at the hardware level so to say. Students want to know this and I think as engineers and scientist I think they should know. If they want to do anything even marginally sophisticated later on okay, if they want to say for example understand parallel computing which is likely to happen.

If they want to get into any kind of high performance computing issues, then understanding low level features is important but for the most part they should not be required to program at a low level. In fact, as far as the programming is concerned, they should do it at a high level and so we should have a programming language which has very high level constructs as well which has high level libraries or which has support for data structure libraries okay.

So that you do not have to build your own data structure libraries, you do not have to deal with low level features such as pointers all the time, pointers or even memory allocation all the time. So we indicated that C++ was a good language which satisfied both of these criteria but of course what we said in the design and many of our recommendations are quite language independent.

Then, as a designer it is important to classify topics into core everything that must be covered as well as what we might call tier 2 core. So it is tier 2 in the sense that it is something that could be covered and it is something that we would like introductory programming courses to teach but we acknowledge that some universities might be forced to have short introductory programming courses in which case the tier 2 material need not be covered.

(Refer Slide Time: 21:20)

Assignments/Exams:

Assignments enable students to

- ▶ Practice/use what they have learned.
- ▶ Learn new applications.
- ▶ Learn practice oriented ideas such as how to design medium sized programs.

Provide sample assignments

Exams:

- ▶ Should be fair: ensure that student has a good path to discovering the solution.
- ▶ Consider giving hints and domain related information as a part of the problem statement.
- ▶ Build upon problems started and half done in class/assignments.

Discuss sample exam questions.

MITTEL

Then, we discussed assignments and exams. Well assignments enable students to practice, to learn new applications as well as we discussed at length and assignments are extremely important for practice oriented courses, practice oriented ideas such as how to design minimum sized programs and the most important, it is very important to provide sample assignments.

So again this is something that may not get provided in sort of very, very short, very, very concise designs of courses but I think that misses the point, I think assignments are extremely important and so course designs should be accompanied by a detailed discussion of what kind of assignments are expected and what role the assignments play and what they should be accomplishing.

Similarly, exams are of course even more important in some sense because students are obsessed with examinations and in a way that is not so bad because the examinations should be indicating what we really think is important and we may talk a lot but what is it that we want students to retain at the end. So that is or that should be tested in the examinations and therefore examination should be fair.

And what that means is whenever we ask questions, we should have in mind how might the students discover the answer and in that explanation that I just gave is hidden one more idea, we do not want examinations that only test if the student is remembering something, we want the student to apply his mind. We do not want a test of memory as far as possible because a test of memory achieves very little.

A student has to apply the ideas in order for the ideas to be learnt and our examinations should be testing application but they should be fair in the sense that we must have done a lot of similar questions or maybe somehow or the other indicated to the student what path the student could take or we must have indicated a variety of paths of which if the student carefully chooses one the answer would be there.

We said that depending upon your students and depending upon your expectations, you should consider giving hints and you should consider giving domain related information as a part of the problem statement. So what this means is that if you ask questions about mechanics or physics, consider giving information about whatever part of physics as a part of the question paper.

Now we do not want to give whole chapters of a physics book, so the ideal situation over here is really that some such physics problems should have been done in class and then we can give very, very brief additional explanation. So may be just state the important law or something like that but we have emphasized that students must be given problems from the domains which are related to their carriers and so asking questions about physics or science is definitely important.

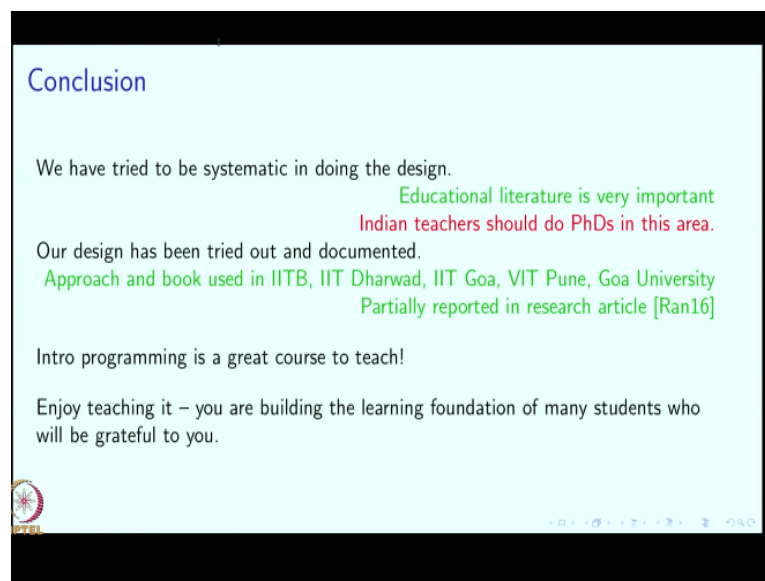
Then, we said that examinations are a very time bound affair and so we cannot be asking difficult questions. On the other hand, we really want students to solve their difficult problems; we want to know whether students can handle difficult problems. So one solution to this is that we will upon what is covered in class. So we have a difficult problem maybe in class we ask an easy version of it.

And then it is perhaps fair to ask the difficult problem in an exam and as a part of the course design, we should discuss sample exams or sample exam questions. Full exams need not be discussed but some kind of indicative questions should be discussed and we should provide the whole range of possible questions that could be asked okay. So I think that is what our strategy was and we believe we have stuck to that strategy to a good extent and now let me make some concluding remarks.

Well friends I have enjoyed teaching this course to you. I hope you have enjoyed learning it. While teaching I have tried to be systematic in doing the design of the course and I should point out at this juncture that the educational literature has helped me a lot in this regard and I feel that all of us all Indian teachers should be referring to the educational literature whenever we redesign courses.

But we should also be contributing to the educational literature. I think we have a wealth of experience. We can make measurements of how our students learn and we will be able to enlighten our own colleagues as well as our international colleagues.

(Refer Slide Time: 27:10)



I would also like to say that the design that I have outlined for you has been tried out and has been documented and in particular the approach as well as the book has been used in IIT Bombay, IIT Dharwad, IIT Goa, VIT Pune, and Goa University and we have reported our experience partially in a research article in one of the prestigious conferences in the computer education field.

I would like to end by saying that I have very much enjoyed teaching the introductory programming course and I hope you will do the same. Please enjoy teaching it, remember that you are building the learning foundation of many, many students who will be very, very grateful to you. Thank you very much.