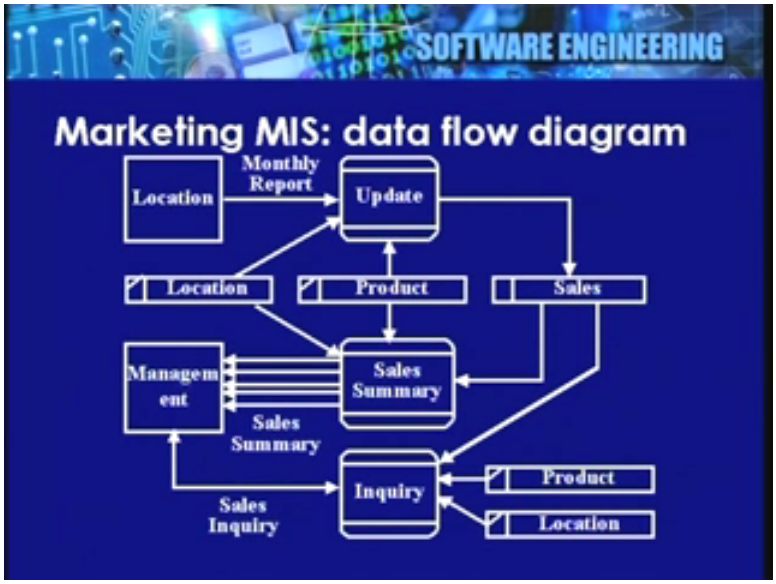
Software Engineering
Prof. Shashi Kelkar
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture - 33
Estimation - II

Let us now continue our session on estimation. In the earlier section we saw what a function point is. The function point is the measure of the size of the software product. To illustrate those some other concepts let us now take an example of marketing [enterprise...1:12]. Imagine for the time being there is a marketing organization which has countrywide operations, it has offices in several cities which are located at different places all over India and each of these particular regions sells a wide range of products through several channels. At the end of the month each of these regions is supposed to send a performance report to the head office. And this report is basically a product-wise sale summary for the month and it is a means of communication between the regional office and the head office. The report is sent to the head office by different regions by different means ranging from e-mail to hand delivery.

Now at head office we are required to summarize this particular data and also generate a database which can be used as a foundation for a limited [decision…2:22.] support system for the top management in the sense that some kind of an enquiry facility can be provided on this particular database. So every month the proposed system will update the sales database and generate some fixed number of reports as required and also provide an enquiry facility for the management's use as and when. Hence five reports and an enquiry facility is the requirement.

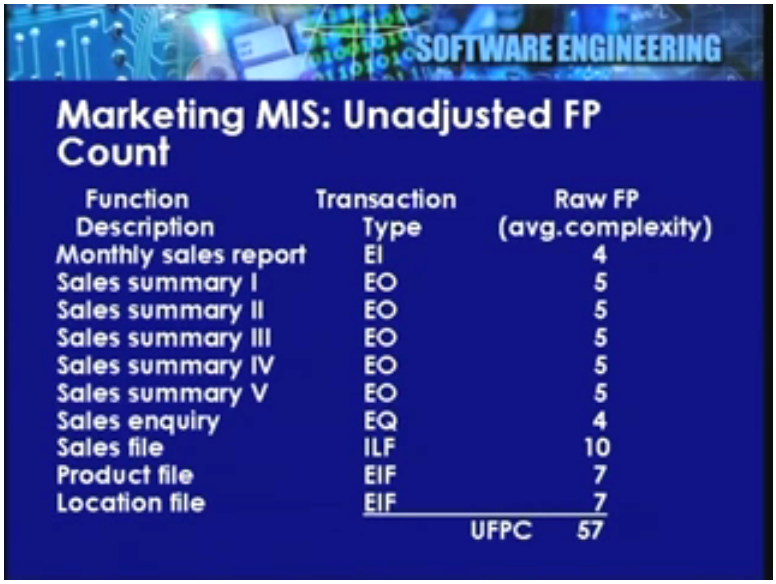
In addition to the sales file that we generate our system will require to use the product file and location file and both these particular files are assumed to be maintained by the EDP department of the company. So it is in this particular context now that you are required to develop a small software and basically give an answer to the current question that we have that is how big is this particular software and obviously in turn how much it will cost and how long it will take to develop this particular software. Now the easiest way to look at it would be as a data flow diagram.

(Refer Slide Time: 3:39)



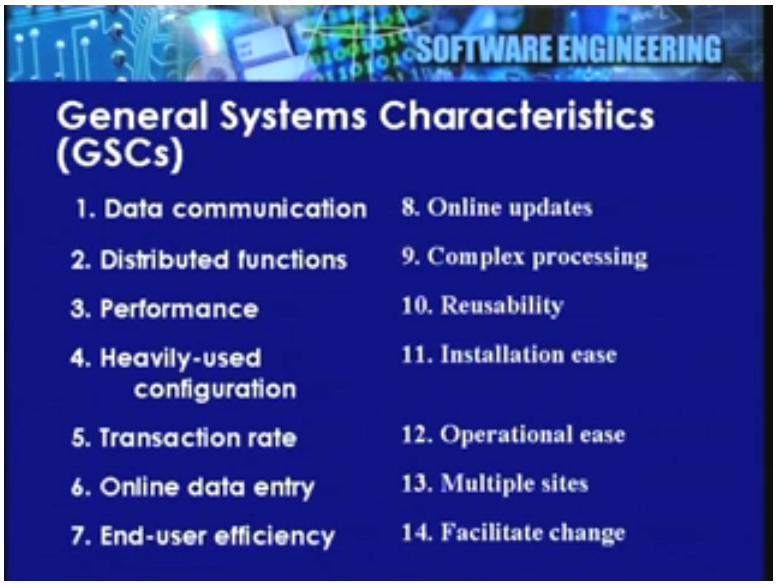
There is a data flow diagram, first level DFD drawn using games and [s……3:54] notations the boundary of the system as implied. So except the two squares which are supposed to be the external entities everything else is inside the system and you have one input and you have five outputs and an enquiry facility, there are three files and this cross [ar…4:!6] indicates that there is a multiple depiction indicating that the file appears only once in the data dictionary though it is shown twice in the diagram. So based on this particular kind of a situation now we were to calculate the number of functions that you have then a simple enumeration can be done. We have a monthly sales report coming in as the input and five summary reports and an enquiry and a sales file as a ILF and a product file and a location file as a EIF.

(Refer Slide Time: 4:57)



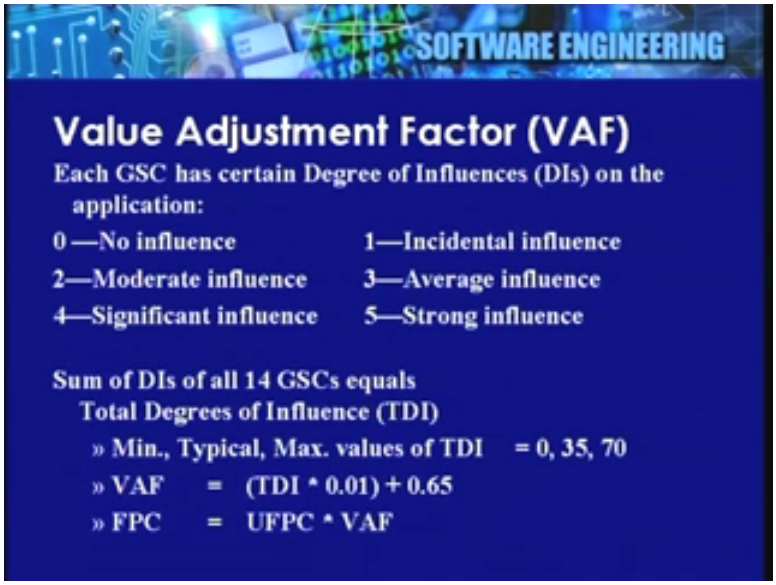
So if you again look at the slide we see all these particular functions mentioned. The type of functions has been indicated here and we have drawn out the average complexity, draw function points for each of these particular functions and this indicates that this system is 57 unadjusted function point count. Now if you were to look at this particular number [….5:33] we need to now convert this unadjusted function point counts into adjusted function point counts by identifying the weightage or the degree of influence for the fourteen general system characteristics.

(Refer Slide Time: 5:49)



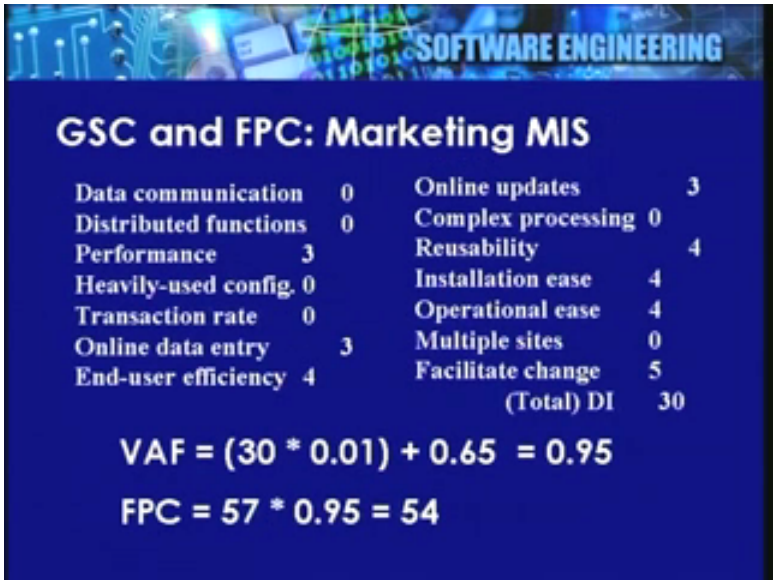
These fourteen characteristics have again been shown on this particular slide. So these range from data communication, distributed function, performance, heavily used configuration, transaction rates, online data entry, end user efficiency and so on as so forth. Now let us see how these particular general system characteristics are influencing our system and each of these particular characteristics will have to be weighted with 0 to 5 as the wait with 0 indicating no influence and 5 indicating the strongest influence on the system.

(Refer Slide Time: 6:18)



Now what will happen is in case you were to do this particular case where all these fourteen characteristics were to be given a 0 influence you would get a total degree of influence of 0 and incase all of them were to be randomly given 5 you will get 70 the average being 75. So, in function point calculations 35° of influence are considered as an average system. So we say that the Value Adjustment Factor is equal to TDI into .01 plus .65 thereby indicating that we give a weightage of 1% to every degree of influence which is in excess or less than the 35 the average degree of influence.

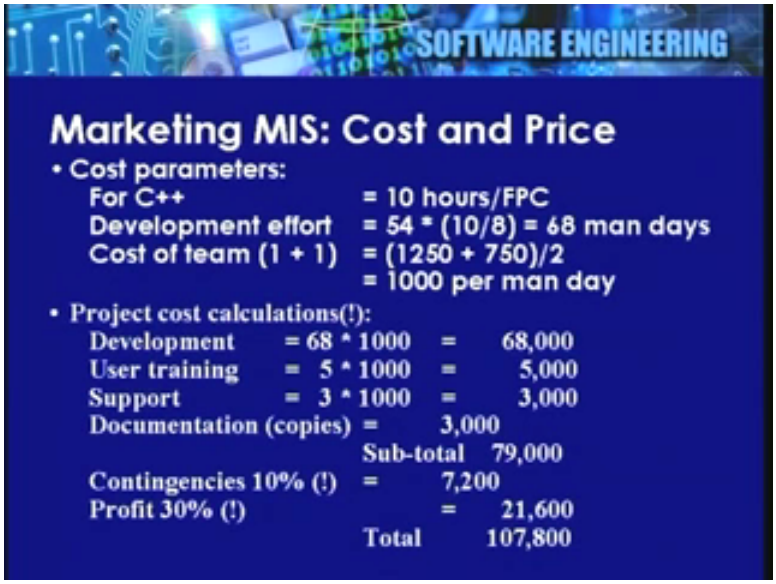
(Refer Slide Time: 7:29)



Therefore based on this situation we can now see that how these particular degrees of influence are as far as the marketing system is concerned. Our system is a PC based stand alone system; it is not a distributed system so the degrees of the influence for these two are 0, performance is somewhat not a very great deal or stringent requirement but some requirement demand is there, it is not a heavily used configuration, the transaction rate is not heavy, online data entry is important and end user efficiency is important because we are not really expecting computer professionals to be using and operating this particular system. The data update is online and we would like to have reusability in mind as that is the way of thinking today. Ease of installation and ease of operation is also very important.

Last but not the least we would like to facilitate change in the sense that we should be able to modify this particular system as and when required. So if we add up all these particular degrees of influence you get a total of 30. So our adjusted function point count is 30 into .01 plus .65 is equal to .95. Therefore, adjusted function point count is equal to 57 that is our unadjusted function point count multiplied by .95 approximately equal to 54 function points. So our system is approximately 54 function points.

(Refer Slide Time: 9:12)



Once we have got this particular data the next thing that we are interested in is to find out how much will this particular system cost. Suppose you have decided that this particular system will be developed in C++. And you have got certain norms which tells you that incase you are using a COBOL kind of a language you will have about 15 hours per function point, any cryptic language like C, C++, FoxPro or dbase or any such particular thing would take about ten hours of function point and if you are using RDBMS and SQL kind of a thing it will take about six hours per function point count. So from our point of view we say that our rate is ten hours per function point count and then the development effort would be 54 function point counts into 10 into 8. So ten hours per function point, eight hours per days gives us the result that the system requirement is 68 mandays of effort.

Now if you assume that the cost of team is, let us make an assumption that we like a team of 2 that is 1 plus 1 is equal to 2 and one a little senior and one a little junior so our average cost of man power is 1000 per day. Then we can calculate the cost of the system development effort as follows. Therefore development is equal to 68 into 1000 is equal to 68000. Now you can add the other components that are associated with the development of the system. User training for 5 days is that much and support is so much and documentation, some extra copies that the client require is so much etc. And this total you need to add of course some continuance and profit.

In many cases profit is not shown explicitly and the profit figures are added in the development cost itself, it is a hidden kind of a number. But just for simplicity we have shown this particular kind of a figure, if you take the total then you can say that we could quote to our client 1,07,800 or approximately some kind of a number as the total development effort for developing this particular kind of a system. Now we can sum up our function point topic, what are the main advantages of function point approach to software sizing?

The first and the most useful one is that FPA approach can be used during very early stages of development when the information available to the developer is very little.

Second the evaluation effort is very small about 1% of the total development cost. The method is very simple, fairly accurate, easy to learn, it is very easy to explain this to the users and get a conformation about the functions from the user so the users can easily tell you what is the EI and what is the EO, EQ and ILF and EIF etc. So this facilitates verification and last but not the least the method being so simple it does not required use of any software tools.

What are the limitations of function point approach?
The biggest limitation is that it does not provide time estimate. Like incase you are required to tell the management how long will it take for you to develop this particular system you have to take recourse to some other method and the FPA method will not tell you how long this particular software development project will take to develop. Next is it does not give you any phase-wise break up, it only tells you the total development effort is so much, it does not tell you how much is per analysis, how much is for design, how much is for coding, testing etc. Therefore you cannot really plan your resources based on this particular estimate.

Another limitation is that this function point approach cannot be used for estimating the development of system software. It is only useful for application software. If you have to develop system software or utilities or some such particular kind of a thing then the FPA approach is not good for that.

Accumulated data for productivity is obviously required in all methods but also in this particular method. Therefore unless you have good productivity you are required to stick to certain industry norms of the kind that we mentioned. And last but not the least it is a very major weakness for function point. It is assumed that there is no change to software development productivity with the size of the project and it is documented enough number of times and in enough number of authors, enough places and as the size of the project increases the development productivity dramatically drops. Starting from that let us now go to the next estimation technique of COCOMO or Constructive Cost Module. COCOMO was original propose by Barry Boehm in 1981, he has also probably published another book called COCOMO II.

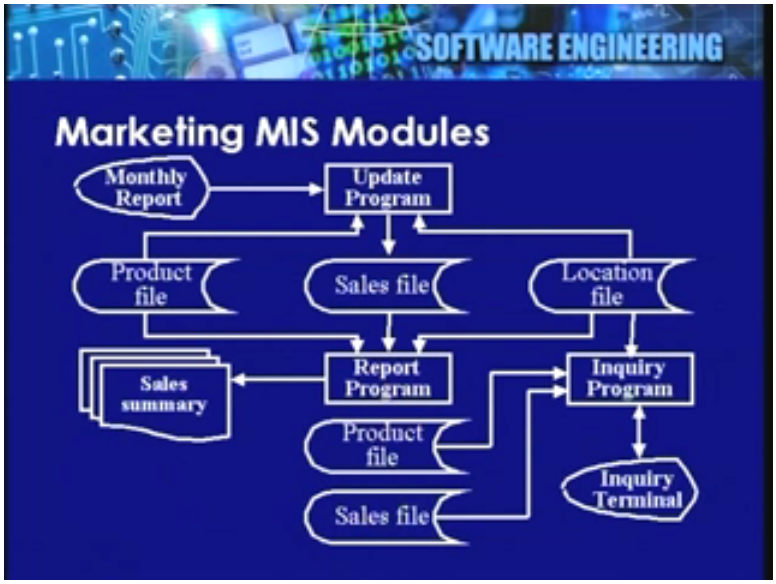
The COCOMO II definitely involves use of some kind of software tools for making estimates. COCOMO is one of the best documented method in public domain. It covers a broad spectrum of software development projects starting from small software to large software and from new development to modification enhancements etc.

(Refer Slide Time: 15:02)

Here when you look in the slide, the COCOMO has three development modes namely the organic, semi-detached and embedded. For our purpose the organic means application software development, semi-detached means development of utility and embedded means basically systems in the developing system software. Also, there is a hierarchy of COCOMO models the basic, intermediate and advanced and the accuracy of estimate goes on increasing as you go on using these particular methods.

Now let us restrict our study to only the basic and the intermediate COCOMO to see how this particular method is useful for making estimates. The biggest advantage is that the method is useful for estimating both the time and the effort involved in developing the particular project but it has a major weakness that it requires lines of code as an input. This means you cannot use COCOMO method for making estimates till you are fairly deep into the project.

(Refer Slide Time: 16:14)



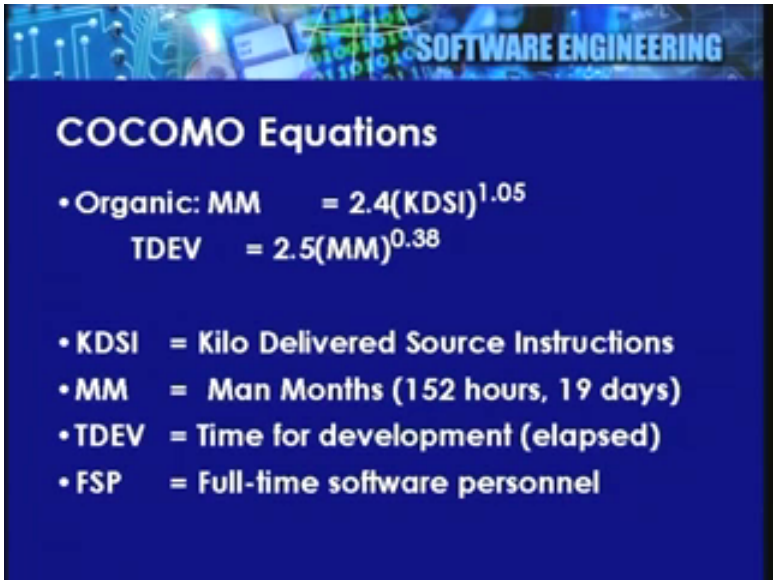
From that point of view using COCOMO will mean that you have to wait to a fairly close to the end of analysis phase. Another particular thing is that the lines of code is a very [d….16:31] term and from that point of view Boehm has describe something called delivered source instruction. In Boehm's parallel the delivered source instruction our DSI sort of give you what are the do's and don'ts or what is in and what is out. For instance, a data declaration JCL formats and libraries will have to be included but number of instructions per line and programming language use and organizational coding standards etc are not taken care of in the DSI. DSI needs to be precisely defined.

So in case you are using COCOMO method then it is better that you exactly follow Boehm's definition of DSI. DSI excludes undelivered supports of testing utilities, drivers, tubs, comments, generated codes, substituted code, reuse code components etc. So it is in this particular context that one needs to be very careful about making sure that you got the right thing in and got nothing else in the DSI that you count. Once we got our initial data assume we have done some kind of a higher level design and you have been able to make a broad structure as to what will and will not be a part of your system. And here you see that we decided that we are going to have three programs; an update program, a report program, this report program will generate all the five reports that we need and there will be an enquiry program. Therefore, from our point of view we are going to have three programs in this particular system.

(Refer Slide Time: 18:31)

Now, using the Delphi and data distribution kind of approach we get a group of people together and ask them how big will the update program be and in this case the group seems to have concurred to say that it will be minimum 250 lines, most likely 300 lines, not more than 400 lines and then using A plus 4B plus C by 6 kind of approach you can say that this particular module will be 308 lines. The query module will be 104 lines and report for all five reports together will be 616, the total is going to be 1028 from our particular point of view and then this will constitute the 100%. Therefore, walking backward you see that your update report is going to be 30%, your query report is going to be 10% and your general reports are going to be 60% of the code. So we look at this particular kind of a data and look at the COCOMO equations now from that particular point of view.

(Refer Slide Time: 19:35)



If you look at the screen you see that there are two basic COCOMO equations. We have only seen the application development organic mode, MM Man Month as we have defined is equal to 2.4 KDSI into 1.05. So Man Month in COCOMO is 152 hours, 19 working days in a month. Please remember a person works for 22 days but 3 days may be occupied in either doing company's work, being on leave, away for training and so on and so forth so useful days out of 22 is going to be only 19 and it is assumed that it is not possible to substitute a software developer for a day on a day today basis as a casual labor. Then KDSI is Kilo Delivered Source Code instructions and if you look at that particular curve you will see that the size of the code goes on increasing, the man month effort will go on increasing more than linearly.

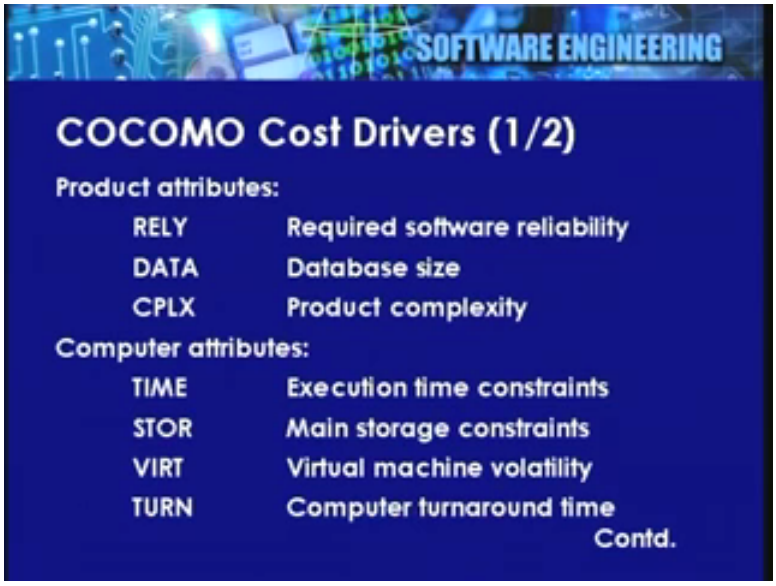
Now, TDEV is the Time for Development and this is given as 2.5 into MM[20:45]. So once you have done the MM calculation you can substitute this figure here and this particular number indicates that the time required for developing the software is less than linearly proportional to the site development effort. Thereby, if you add 100 man month effort then 200 man month effort then the time to delivery for a 200 man month effort will not be twice as much as the time to delivery of the 100 man month effort.

Once you got this particular data that is once you find the total number of man months etc using MM by TDEV you can get something called full-time software development personnel as the measure or the project indicating that how many people we really need to deploy on this particular project. If you were to have this data we need to now again do some adjustment factors.

First let us look at the adjustment factors and then see how these are used. COCOMO divides the adjustment cost, divides the cost drivers into four groups. The product attributes, the computer attributes, the personnel attributes and project attributes. The product attributes, the computer attributes and the personnel attributes to a large extent

are not under the control of the project manager whereas the project attributes are definitely within the control of the project manager.

(Refer Slide Time: 22:22)

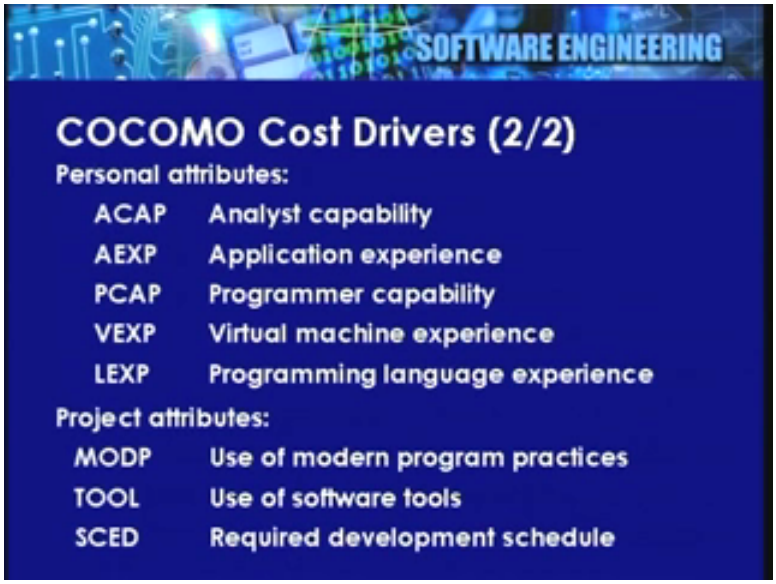


Therefore let us look at what are the basic cost drivers. If you look at the slide first we have three product attributes the required software reliability, the database size and product complexity. You can remember our friend Kalpana Chawla, incase you had to do a development for re-entry of a space capsulin to the earth's atmosphere then the reliability is required being very high. Then here you have a situation where the size of the database to the size of the code they definitely bear some kind of a relationship.

Product complexity:
In case you require a code something like Danzig's algorithm or Karmarkar's algorithm for linear programming from logic is fairly complicated from your particular point of view. The second group of attributes is the computer attributes. This pertain to the machine you are going to be using, the environment so the execution time constraints.

In case if you require performing this particular job in certain areas, the main storage constrains, the virtual machine. the virtual machine volatility means that the complete environment the hardware and the software together, if you are used to developing software under a particular environment hardware then you will find that the subsequent development under the same amendment will be easier. Computer turnaround time is more appropriate for basically the bad jobs but we really do not come across too many of these things in recent applications.

(Refer Slide Time: 24:05)

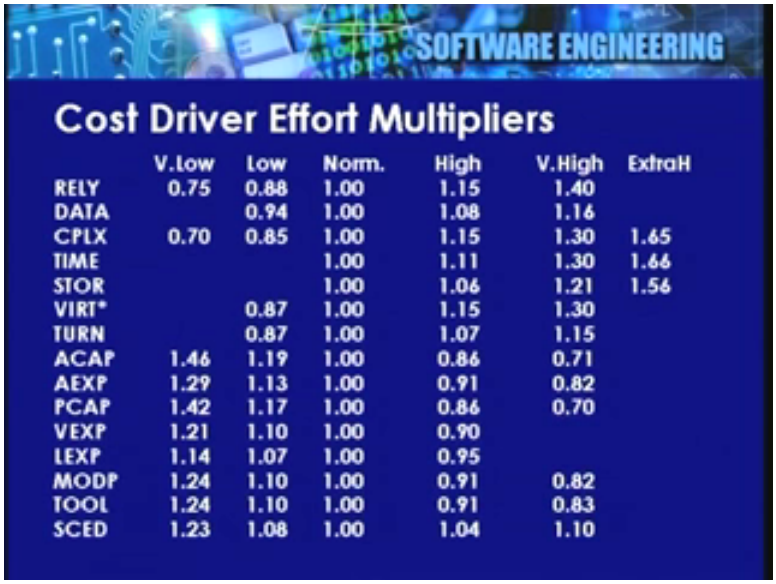


Personal attributes is the next particular thing that you have; analyst capability, application capability of the analyst as to whether the analyst is familiar with the area in which he has been. Similarly, the programmer capability and the programming language experience and of course the programmer if he is familiar with the entire working of the system on which he is going to do the development then that would definitely be an added advantage for the project manager. The three adjustment factors which are under the control of the project attributes are use of modern programming practices.

Just by being systematic and using standard methods and structured approaches etc a very dramatic improvement in productivity is possible. Your productivity can improve further by use of software tools and that is a very interesting particular schedule. There are two types of situations that you encounter in real life. One is you are pressurized, do the job as fast as possible so you want to schedule compression and there is some situation where you would like the schedule to be elongated.

Take a simple example; if a company has an in-house job being done and suddenly receives an order for doing the customers paid job it is likely that the in-house development will be put on the back banner and the project manager will be ask to go slow. So in this particular manner if you were to look at it now there is the effort.

(Refer Slide Time: 25:43)

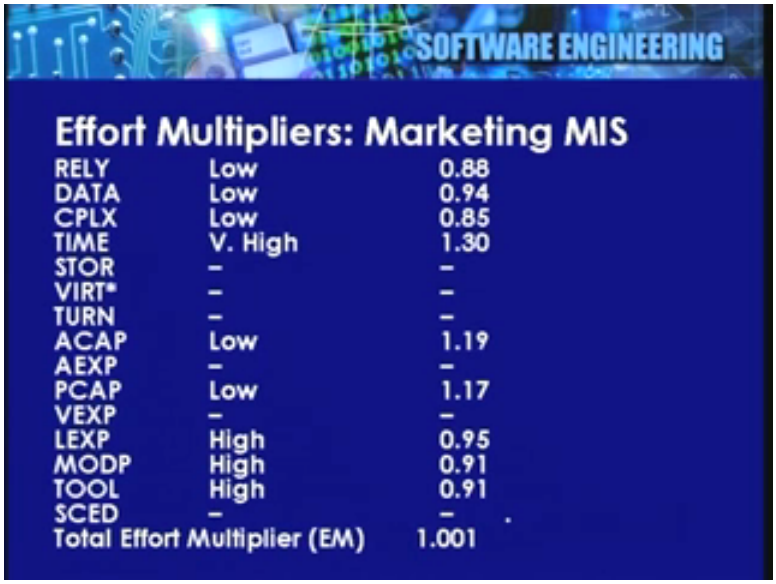


	V.Low	Low	Norm.	High	V.High	ExtraH
RELY	0.75	0.88	1.00	1.15	1.40	
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT*		0.87	1.00	1.15	1.30	
TURN		0.87	1.00	1.07	1.15	
ACAP	1.46	1.19	1.00	0.86	0.71	
AEXP	1.29	1.13	1.00	0.91	0.82	
PCAP	1.42	1.17	1.00	0.86	0.70	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.83	
SCED	1.23	1.08	1.00	1.04	1.10	

We have the effort distribution multipliers for the cost drivers. So we have all the fourteen factors here and for the normal kind of a thing you see all of them. Now this is very different from FBA. Remember, the adjustment factors in COCOMO are multiplicative in nature. So you multiply one into another into another into another and so on. So that way if you anyway multiply fourteen numbers into each other it can have a very dramatic effect on the total adjustment that you are doing to the system.

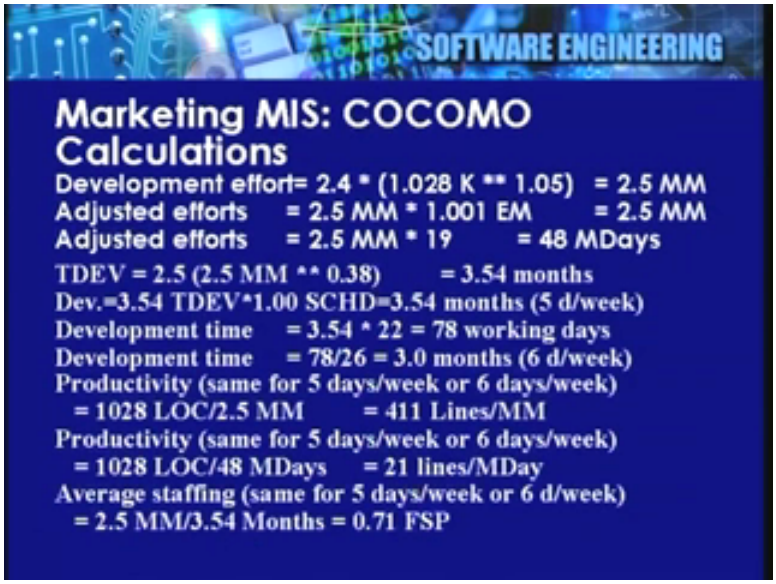
There is another interesting thing. If you look at all these particular factors they seem to go in only one direction from say .075 to 140 except the schedule goes from 123 to 1 and then again increases. What it means is even if you are doing a job slow then the development efficiency drops. It also tells you that there is a lower bound for compressing the schedule to approximately 75% of the estimated time and there is also upper bound for elongation of a particular project to about 60% of the estimated time.

(Refer Slide Time: 27:06)



In this particular month we can get the adjustment factors. Again when we see the marketing example, we can see that wherever we are affected with this particular kind of a thing from our particular point of view the complexity is not very high, the data is not very high but the turn around time may be very critical and you may have the analyst capability as low and programmer capability as also low but the programmer is use to working in the environment you are taking about and we have a very disciplined project manager and from that particular point of view you have very high scores. There is no extra ordinary demands have been made schedule-wise on the developer. So we are doing both using modern programming practices and using tools so from that point of view your development efficiency. So you multiply these numbers into each other you get a number 1.001.

(Refer Slide Time: 28:18)



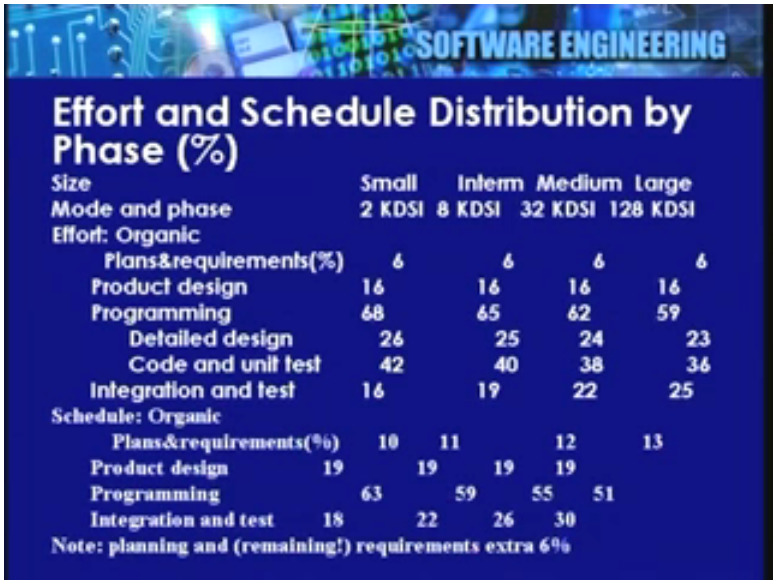
Therefore in case now you have to use this particular data and do the calculations then look at how it is. The development effort comes to 2.4 into 1.028 K raised to 1.05 is equal to 2.5 MM man months. The adjustment effort is 2.5 into 1.001 is equal to 2.5 MM man months. And the adjusted effort in terms of man days comes to 48 man days of effort.

The TDEV you can calculate 2.5 and substitute this particular 2.5 man months here, it is only a coincidence that these two figures are the same but they refer to two different data and you get an answer that the total time to deliver will be 3.54 months, assume that you have 22 days in a month the development can be completed for so many months assuming you are working five days per week.

In case your development is to be done six days a week then you can convert 3.54 into number of working days and then divide by the number of working days. Therefore you say that this project can be delivered in three months provided if you are working six days a week. So in this particular manner we get through the first particular part of COCOMO and we were one up on FPA we were not only able to develop and make an estimate for the development effort but we also tried to say that how long this particular development will take.

Now we come to the next part of COCOMO. We would like to break up this particular time into phase-wise development effort and schedule. So if you have to look again at the slide you have COCOMO equations which tell you that how much percentage of the development time will be attributed to what phase.

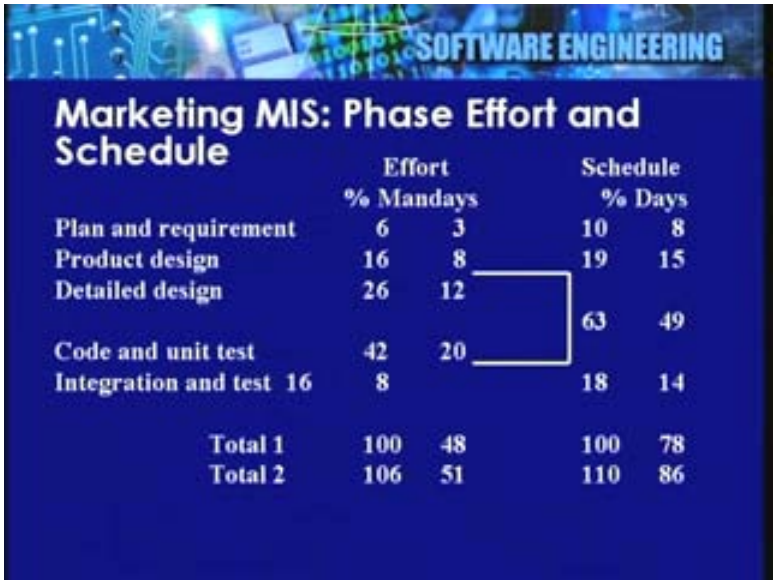
(Refer Slide Time: 30:32)



In case you had a small project, ours is a small project less than 2 KDSI so the product design is going to take 16% of our product development time, the programming is going to take 68, if you want you can break it up into detailed design and coding and unit testing as 26 and 42 and then the integration and final testing is going to take 16% of the time. You might need an additional 6% time for the planning and the remaining part of analysis that may be required. Similar data is available for the other types also. You must realize that as the size of the project increases the coding and unit testing percentage goes down whereas you will find that from your particular point of view the integration and test percentage will dramatically go on increasing.

On similar lines you also have the beak up that you are going to get for schedules. And the schedules that you have here take 19%. Now only one difference is the break up for the programming is not given in this particular schedule break up that you here.

(Refer Slide Time: 32:00)



Now using this particular data if you were to now look at our marketing example you can easily say that the percentage mandate as seen from the earlier table is given here and the total effort is given here in terms of days. So 6% of the total development effort is going to be 3 it is 48, our development effort is 48 days and 6% of that approximately is 3 so it is going to take 3 days of effort for doing the remaining planning and requirement, 8 days for doing product design, 12 days for detailed design, 20 days for coding and testing and little offset here and you will require 8 days of effort for you integration and testing. So your total 100% development effort is 48 and of course including the additional effort that you require for planning and remaining requirement is 51%.

Therefore, in this particular manner we can do the job for effort and on exactly similar lines we can do this particular effort for schedule. Therefore you see here that we got the particular percentages of the total time required for doing the development and you multiply by the time to develop that is 78 days and you get so many days for doing it. So we can summarize and say that your product design for instance is going to take 8 days of effort spread over 15 days of time and your programming is going to take in 32 days of effort over 49 days of time. And last but not least your 8 days of integration effort is going to be spread over 24 days.

Let us not re-look at how we use the COCOMO approach. To begin with we did a fairly deep work into our system to do a higher level design that is after getting the requirements we identified which are the main programs that we are going to write and we came out with some kind of a run chart from which we could easily say that we are going to have three programs; the update program, the report program and the enquiry program. Once we got this particular thing then we use the combination of Beta and Delphi technique assuming that we had more than one expert required to do the estimation. Actually this particular system is too small to required more than one expert.

In case you required you could use it using Delphi approach or wide band Delphi or any such approach or may be by using a single persons estimation we used the Beta distribution kind of approach and we made minimum, maximum and the most likely estimates in terms of the number of lines of code or delivered source of instructions for each of the three programs the update program, report program and the query program.

Once we got this particular kind of a thing the next particular thing was to plug the number of lines of code into the COCOMO organic equations to get the effort estimate for man months and to get a TDEV the Time to Develop estimate and from that also make an estimate for the full-time software personnel required for doing this particular kind of a job.

The next step was we had adjustment factors, the product attributes, the computer attributes, personnel attributes and the project attributes. We ranked our system on each of these particular attributes and multiplied by the cost driver effort multiplier that was given in a standard table and came out with a total adjustment factor noting that the adjustment factors are multiplicative in nature. In that particular sense we got a number and once we had that particular thing we could calculate the unadjusted man month effort for developing this particular system.

Now we also saw that man months could be converted into man days and the TDEV in months could be converted to number of working days. We could manipulate these particular working days by working either five days a week or six days a week and you could possibly think of working seven days a week also, you can think of working 1 ½ shifts or 12Hr shifts etc. But basically remember that the working days have to be the anchor for our calculations.

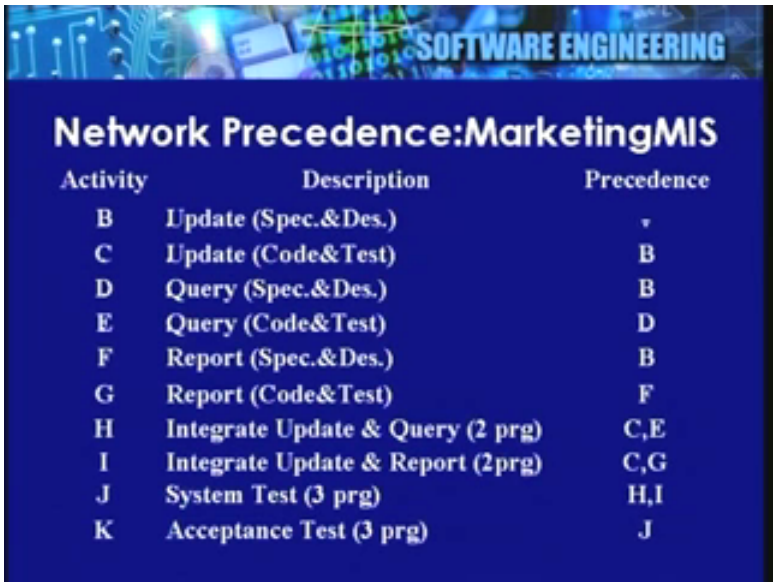
Doing this particular thing we will see that the development production will remain the same irrespective of working five days a week or six days a week and our development productivity turned out to be about 411 lines of delivered source coded instructions per man month of effort. Thus with the conclusion that we can now assign work to different groups of people at different point of time we could go ahead. In case you were to use the next level of COCOMO you can also get activity-wise break up for each development stage.

The next particular part is phase break up and the distribution by phase is given for different tables, there are different tables available, we saw only one table for organic model but similar tables are available for semi-detached and embedded models and it gives you a break up for the total development effort and the total development schedule in terms of the product design, the programming effort and integration and testing effort. So doing these kind of things you could come out with a conclusion and say that how long will the system take to develop and how much analyst effort will be required, how much development effort will be required, how much coding effort will be required and approximately when you will require this particular kind of a thing. This data will be very useful for the top management for allocating resources to your project.

Let us now divert our attention to the third technique, your use of network analysis for estimating the time required and the effort required for individual programs in the system.

A very quick review of steps involved in network analysis:
We are very familiar with things like work break down structure. But first you need to enumerate the list of activities. Once you got the list of activities you need to draw precedence table, you also know what precedence is, then you can draw an activity diagram, estimate the early, likely and late times for each of these particular activities so you can use like Beta distribution kind of approach. once you got this mean time required for doing each of these particular activities you calculate six different times associated with the system as what you call the earliest even time and the latest even time associated with this [even….39:35] and latest starting time, earliest finishing time and latest finishing time with each activity. Once you got this data you can easily calculate the slacks and the flouts, the slacks for the events and flouts for the activity time that you have and you would get a critical path. From the critical path you could always allocate resources and then convert the network into a scheduling network and draw the corresponding resources histogram.

(Refer Slide Time: 40:17)



Now how will you go about doing this in our particular project?
We start with our three programs and giving a little twist to the particular matter let us assume that each of these particular programs can be done in two parts. First particular part is to do the specification and the design and the other is to do coding and testing. Many times you find that coding and testing and the specification writing are dependent on each other but the subsequent programs may only depend on the specification part of the earlier program and not necessarily on the coding and testing of that particular program. So we are now sure that we have two parts of the update program and we have another two parts for the query program and we have another two parts for the report program. So three programs have been split into six activities, their precedence

relationships have been identified here and then we do the integration. Now the designer will have to decide how to integrate the system.

But from our point of view let us assume for simplicity sake that we have decided that we will do the integration in steps. First we will integrate the update program with the query program and then the update program and the report program and then we will integrate all of them together at the time of system testing and of course the acceptance testing. Now here we need to make some assumptions.

What kind of assumptions we need to make?
Let us make an assumption that the amount of time required for doing integration is directly proportional to the number of units or programs being integrated in steps. That means if you are integrating two programs the effort required will be two units and in case you are integrating four programs at the same time the effort required will be four units of time.

Now, making this particular assumption that each stage of our testing we can put on how many programs are really going into each of these particular stages. So in our case we identify that for the system and acceptance testing we are putting three programs together and during the update enquiry integration you got two programs and update and report you have got two programs. Therefore these are all the entire precedence that we have got within these particular activities.

Now looking at this particular data we can now bring up our COCOMO tables and see how we can get a detailed break up of that. We have now regrouped the data, we put the entire specifications together update query and report together and we have update query and report for coding and testing together and then the integrations still remains where we are and the precedence is given.

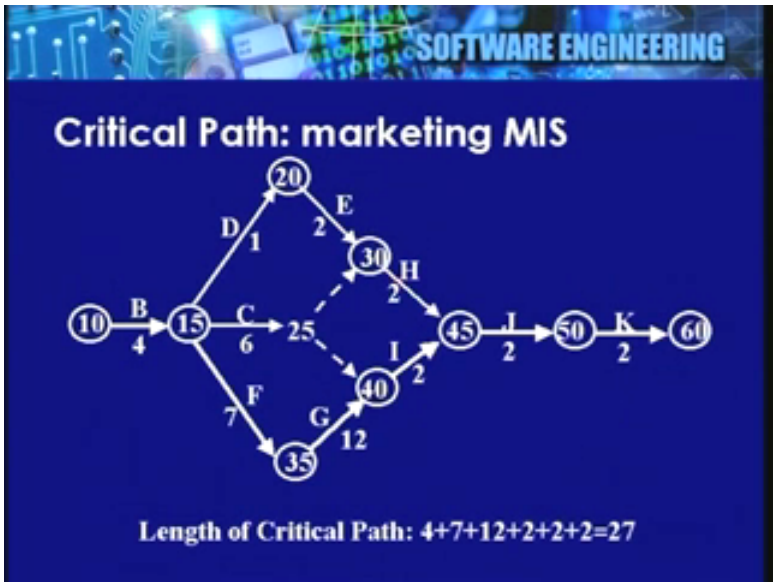
We have seen from the earlier COCOMO kind of an exercise that the phase effort involved in writing these or doing these three parts the specification for update query and the report program together is going to be 12 days and the programming effort and coding effort is going to be for 20 for this particular activity and then for integration all the integration and testing put together is going to take 8 days. We also know that within this particular break up the weightage of these three programs is 30%, 10% and 60% and 30%, 10% and 60% here also and we have made the assumption that the total number of programs integrated in the steps that we have will determine the break up of the integration time.

So from our point of view if you look at the testing time the first two integrations required two programs each and the next two integrations required three programs each. So the total number of programs that went into such testing steps of integrating and testing together is ten programs. So we say that the total effort involved for doing the testing can now is broken up, the integration and system and acceptance testing can be broken up in this particular proportion.

Here in the slide you see that you have made this 2 by 10, 2 by 10, 3 by 10 and 3 by 10 as you see.

Now when you multiply 12 by these numbers of course the corresponding numbers are the percentages you get an activity effort. Now I have rounded off all the numbers to approximately not more accurate than one day and from that point of view it might have some kind of an apparent anomaly in terms of all the four activities getting two days each.

(Refer Slide Time: 45:20)



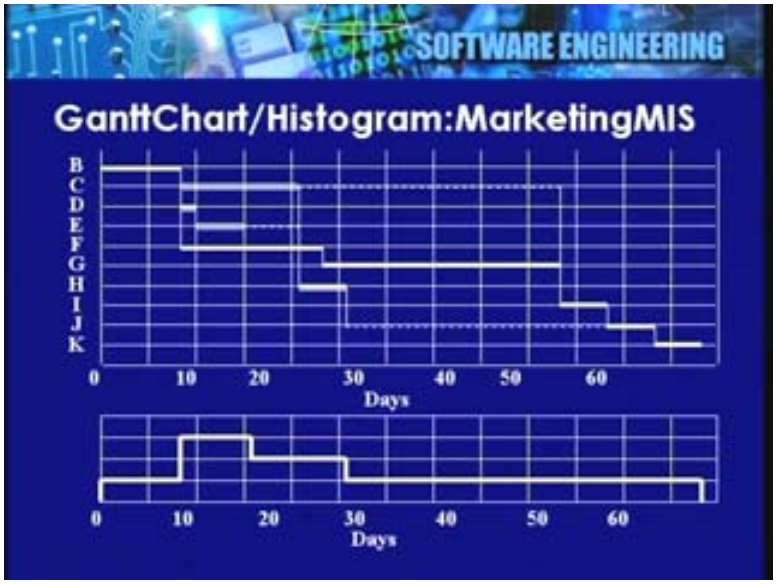
So we got this data and now we can easily draw a network. So here we draw a network for our software development coding and integration testing kind of effort. We got all the activities, we got dummy activities and if you look at this particular thing then you find that the critical path that we have is only 27 units. Here we are making an assumption that the amount of effort that you have is the equal number of days that you are spending but that really is not going to be true.

Now we have seen that this effort from the total time elapsed time for completing the project that is available at our disposal from this point to this point is 78 days. So what we need to do now is take this particular data and break it up into some kind of accordion so that we multiply all the elapsed time calculations by 78 by 27 which is the total critical time that we got which gives us the factor of 2.33 and now if you look at this particular slide you will find that the update is going to be 4 into .033 and this is going to give you the elapsed time for completing each activity.

So remember, the earlier times that we got, this particular thing gives you the effort involved in completing the activity, this gives elapsed time in completing the activity therefore identify as to which activity is critical and in case you have to compute the critical path you find that this particular number is fairly close to what number we are

supposed to get from our earlier COCOMO table that is 78 minus the data that is associated with the earlier part of doing the remaining part of analysis and so on and so forth. Thus we come to the conclusion that the time required for doing the job and the effort involved in doing the job is shown there.

(Refer Slide Time: 47:25)



So we go to the next particular slide now and we can draw this data as the Gantt chart. Here you see that over a time period of time we have indicated which particular activity it is, when it is going to start and when it is going to finish. There is of course certain amount of slack associated with these particular activities and then in case you were to do as early as possible kind of a schedule then we will get a histogram of this particular type as shown below. This is how the break up will go for doing the program development effort.

Now from our particular point of view we will look at it and say that how does this help us in completing the job. We did three things. To begin with early in the game we used FPA technique and made an over all effort estimation for the system. Then we got the job and then we did some kind of a broad estimate, some broad higher level design and estimated the number of lines of code for the system and then used the COCOMO method and went to management with a plan to indicate how much resource of what type will be required approximately when. Then when we got the resource we only talked about the program development effort then we were able to say that we have so much time availability in disposal and we are required to deliver the project in so many days and there are so many programs.

We decided what is going to be the development strategy, what is going to be the integration effort actually the integration pattern then the integration effort and we were able to come out with the numbers for the time required for completing each activity and then using the PERT CPM approach we could easily come out with the time required for

starting and finishing the individual program and by making sure that the total development time was not contradicting those calculated by the COCOMO effort.