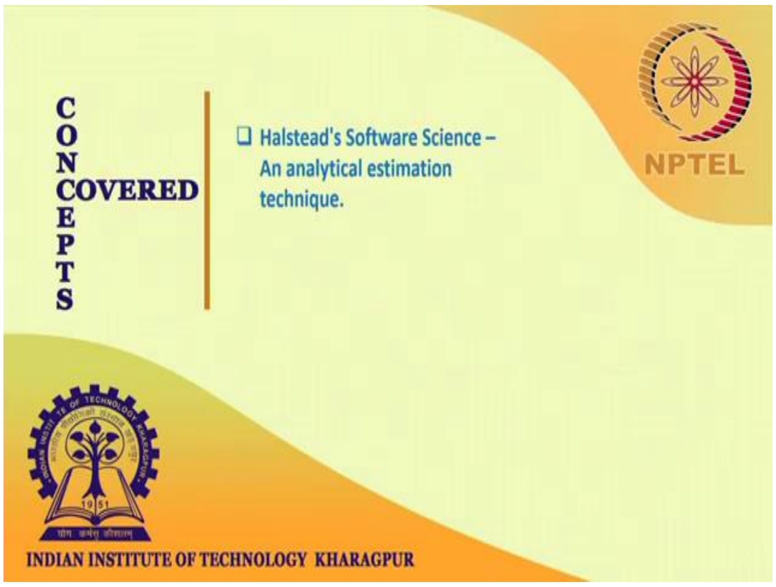
Software Project Management
Prof. Durga Prasad Mohapatra
Department of Computer Science and Engineering
Indian Institute of Technology, Rourkela

Lecture – 24
Project Estimation Techniques (Contd.)

(Refer Slide Time: 00:22)



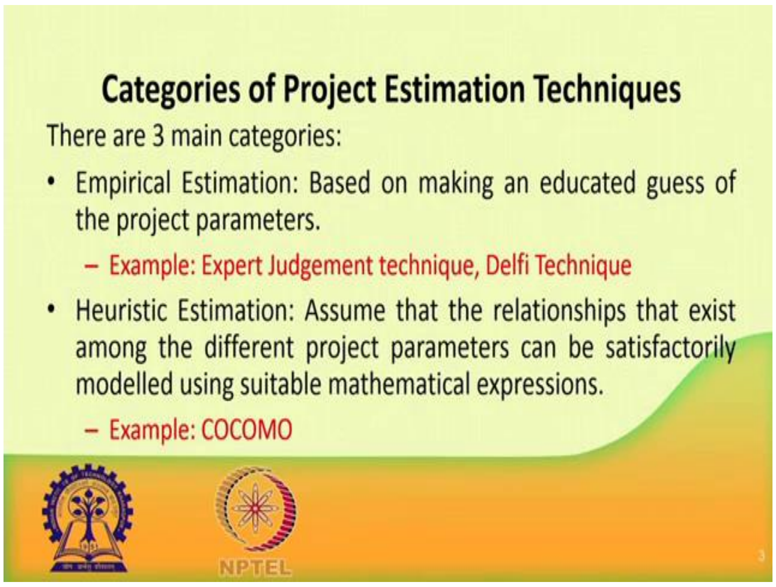
CONCEPTS COVERED

- Halstead's Software Science – An analytical estimation technique.

NPTEL
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Good afternoon. Now, let us take another kind of Project Estimation Technique that is Halstead’s Software Science which is an analytical estimation technique.

(Refer Slide Time: 00:30)



Categories of Project Estimation Techniques

There are 3 main categories:

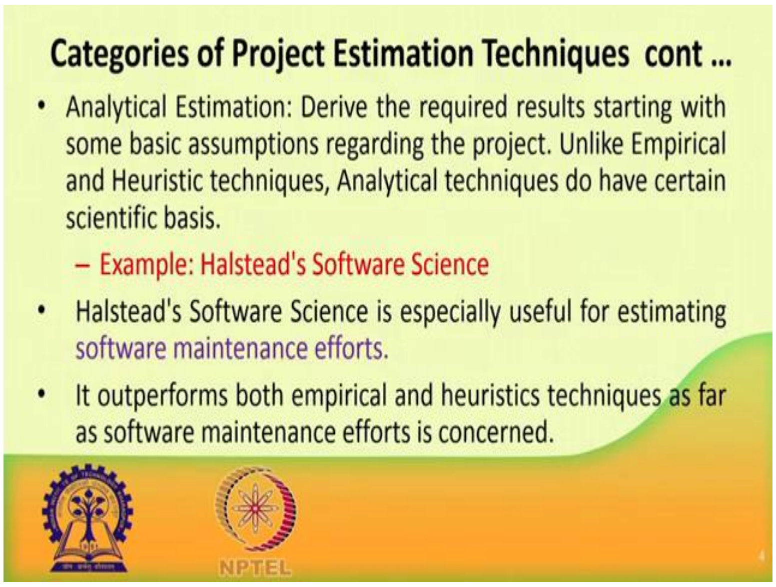
- Empirical Estimation: Based on making an educated guess of the project parameters.
  - Example: Expert Judgement technique, Delfi Technique
- Heuristic Estimation: Assume that the relationships that exist among the different project parameters can be satisfactorily modelled using suitable mathematical expressions.
  - Example: COCOMO

See, in the in the while you have started the project estimation technique, now we have known that there are 3 main important categories of testing estimation techniques, number one empirical estimation, then heuristic estimation and analytical estimation. We have already discussed empirical estimation and heuristic estimation. Normally, empirical estimation is based on making an educated guess of the project parameters. In a project parameter, in a project, there are different parameters such as size etcetera.

In empirical estimation first, the project manager makes an educated guess of the what the project parameters and then gradually they take a decision they estimate the various what other parameters such as effort, costs etcetera. So, here basically this technique based on making an initial educated guess of different project parameters and then they calculate and refine what are the other what parameters. So, examples of empirical estimation techniques are expert judgment technique and Delfi technique. We have already discussed expert judgment technique and Delfi technique in the previous classes.

So, next category is heuristic estimation. So, this technique assumes that the relationship which exists among the different project parameters can be satisfactorily modelled using some mathematical expressions ok. So, here this estimation technique it is based on the what it assumes that what are the relationships they exist among the different project parameters they can somehow be modeled, they can be expressed using some suitable mathematical expressions. And then you can then later on you can refine them by using some what multipliers or cost drivers in order to get the refined or more accurate estimates. The examples are COCOMO models, we have seen different kinds of COCOMO models under this heuristic estimation such as basic COCOMO, intermediate COCOMO, complete COCOMO or detailed COCOMO and COCOMO 2. So, all these things we have seen in the previous classes.

(Refer Slide Time: 03:09)



Categories of Project Estimation Techniques cont ...

- Analytical Estimation: Derive the required results starting with some basic assumptions regarding the project. Unlike Empirical and Heuristic techniques, Analytical techniques do have certain scientific basis.
  - Example: Halstead's Software Science
- Halstead's Software Science is especially useful for estimating software maintenance efforts.
- It outperforms both empirical and heuristics techniques as far as software maintenance efforts is concerned.

Then next category is analytical estimation. Analytical estimation is based on the following concept. In this technique, the project manager he derives the required results how starting with some basic assumptions. So, assumptions on what, assumptions on regarding the different what a project parameters, so basically here the parameters such as efforts and size or these what you can say program volume, etcetera, they are derived starting with some basic assumptions regarding the different project parameters. But unlike the empirical and heuristics techniques that we have already seen, the analytical techniques they have certain scientific basis.

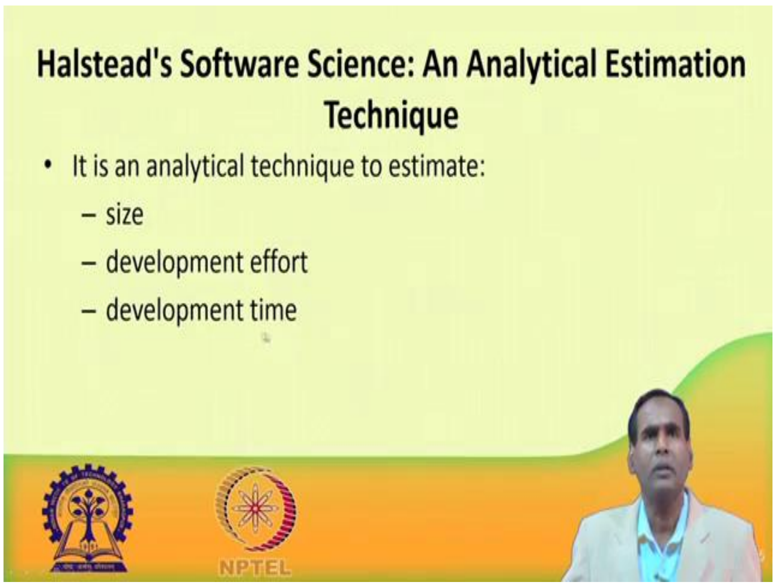
So, these techniques they have certain scientific basis. So, these assumptions are not vague assumptions. So, all those what assumptions or the different equations or the different methods that will be used in these techniques they have some certain what scientific basis. And under this category, you will see one such example is coming under analytical estimation techniques that is known as Halstead’s software science. So, today we will discuss about the Halstead’s software science how it can be used for different for the estimation of different project parameters.

Halstead’s software science is especially useful for estimating software maintenance effort. After the software is put into use after it is deployed, so next phase is maintenance phase. So, during maintenance how much effort how; much effort has to be given, so those things can be easily estimated by using Halstead Software Science. So, especially

Halstead software science is very much suitable for estimating software maintenance efforts.

This technique outform outperforms both the empirical and heuristic techniques as far as software maintenance effort is concerned. So, as a as far as the software maintenance input is concerned, it gives better results. It more accurately estimates the different parameters such as effort and cost etcetera. It performs better results than the previous two techniques we have seen empirical and heuristic techniques.

(Refer Slide Time: 05:49)

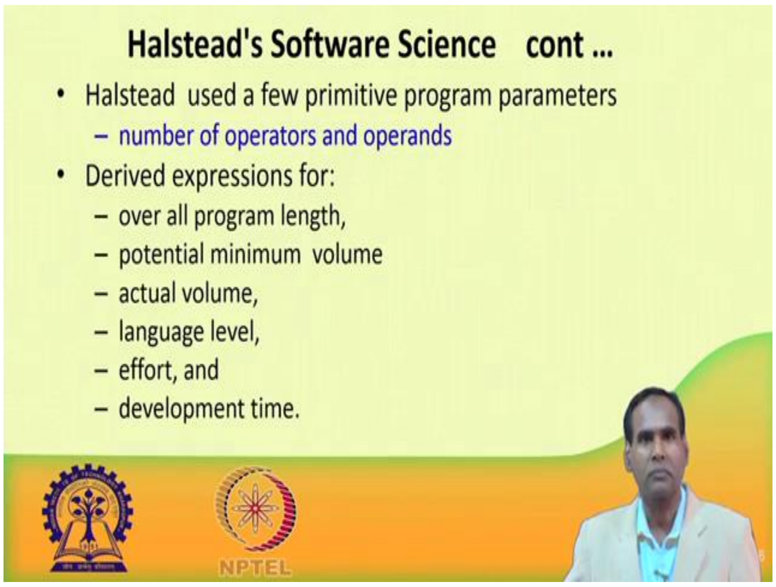


Halstead's Software Science: An Analytical Estimation Technique

- It is an analytical technique to estimate:
  - size
  - development effort
  - development time

I have already told you that Halstead software science is an analytical technique for what to estimate different parameters of a project, such as the size, the development effort, development time, the program volume, and so many other things you will see for what parameters it can be used to estimate.

(Refer Slide Time: 06:15)



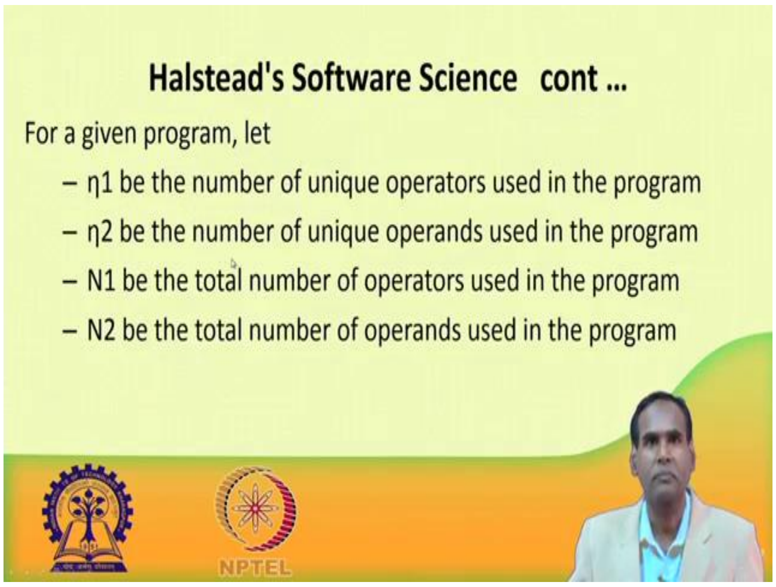
Halstead's Software Science cont ...

- Halstead used a few primitive program parameters
  - number of operators and operands
- Derived expressions for:
  - over all program length,
  - potential minimum volume
  - actual volume,
  - language level,
  - effort, and
  - development time.

Halstead, he has used a few primitive program parameters in order to estimate effort, cost etcetera. So, what primitive program parameters, he has used for the estimation purpose? He has used two important, what a primitive program parameters, one the number of operators and two, the number of operands. So, by using the number of operators and the number of operands, any manager - program manager, can estimate certain parameters for each project.

So, what parameters can be estimated? The following parameter can be estimated. Using Halstead software science you can derive expressions for estimating the overall program length, the potential minimum volume, the actual volume of the program, the language level of the program, the effort that will be spent to develop the program and finally, how much development time will be required to develop that program, so that also can be estimated. So, Halstead software science provides or by using Halstead software science, you can derive the expressions for these parameters for estimating these parameters.

(Refer Slide Time: 07:44)



Halstead's Software Science cont ...

For a given program, let

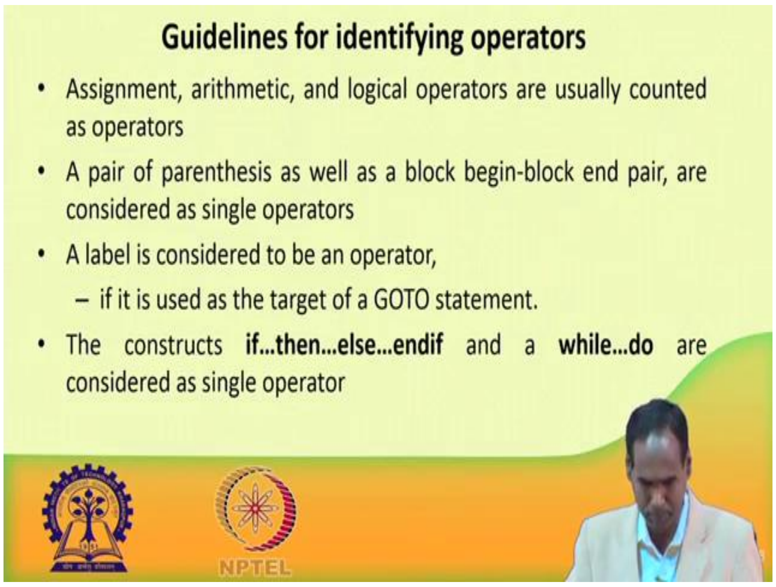
- $\eta 1$ be the number of unique operators used in the program
- $\eta 2$ be the number of unique operands used in the program
- N1 be the total number of operators used in the program
- N2 be the total number of operands used in the program

Now, let us see how software's Halstead's software science works. For a given program let eta 1 be the number of unique operators which are used in the program, eta 2 be the number of unique operands which are used in the program, N 1 be the total number of operators used in the program, and N 2 be the total number of operands used in the program.

So, basically eta 1 and eta 2 are unique operators under unique operands, whereas N 1 and N 2 these are total number of operators and operands used in the program. With this we can derive the expressions for these parameters what we have shown in the previous slide such as overall program length, potential minimum volume, actual volume, language level, effort, and development time, etcetera.

Now, let us see how by using these terms how we can estimate, how we can derive expressions for the desired parameters such as volume and the program length etcetera. Before going to derive the expressions, let us first see some of the guidelines that will help us to identify the unique operands, the unique operators etcetera.

(Refer Slide Time: 09:02)



Guidelines for identifying operators

- Assignment, arithmetic, and logical operators are usually counted as operators
- A pair of parenthesis as well as a block begin-block end pair, are considered as single operators
- A label is considered to be an operator,
  - if it is used as the target of a GOTO statement.
- The constructs **if…then…else…endif** and a **while…do** are considered as single operator

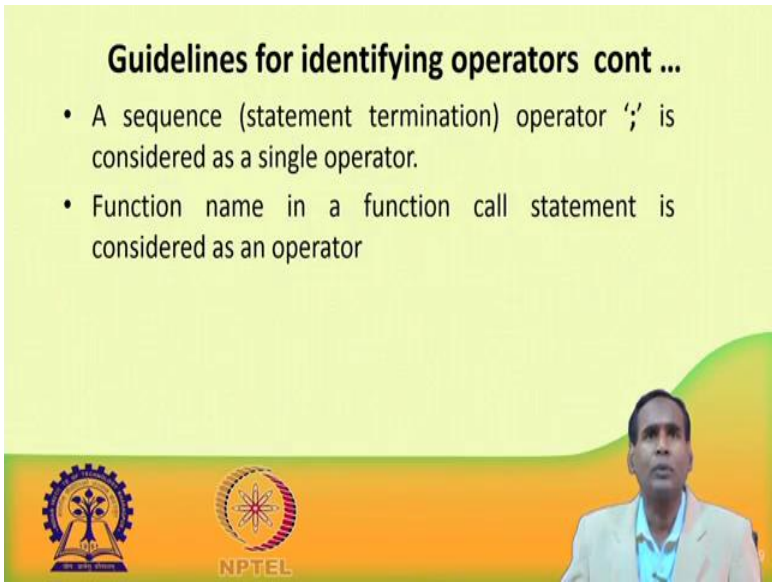
These are the guidelines they may be used for identifying the operators. It says that assignment operators, arithmetic operators and logical operators, they are usually counted as operators. So, all such assignment operators, arithmetic operators such as plus minus etcetera, and logical operators etcetera they are they can be used, they are usually counted as operators.

Similarly, in a arithmetic expression where parenthesis is there. A pair of parenthesis as well as a block begin and end pair are considered as single operators. So, if there will be a pair of parenthesis or a block starting with a begin end block, then that will be also what sorry begin block and the end block that pair will also be considered as single operators not two operators.

Similarly, if you are using labels, a label is also considered as an operator label is considered to be an operator if it is used as the target of a GOTO statement. So, if you are using a GOTO statement, and the target is a labelled, then also label is are considered as an operator. The construct such as if then else endif, and a while do, do while etcetera, they are also considered as single operators.

A sequence such as the statement termination operator; a sequence operators such as the statement terminator that you are using the semicolon very often you might have seen in C programs etcetera that end with a semicolon these statements end with a semicolon. So, here the semicolon operator is also considered as a single operator.

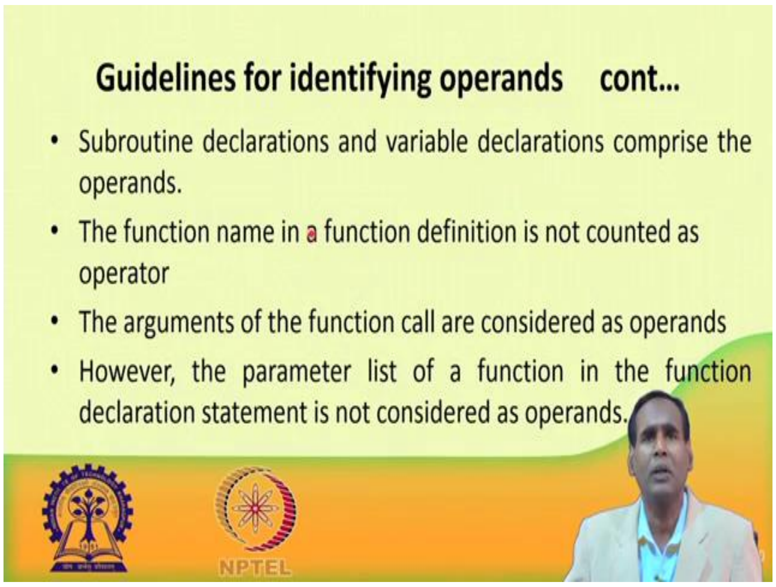
(Refer Slide Time: 10:25)


Guidelines for identifying operators cont ...

- A sequence (statement termination) operator ';' is considered as a single operator.
- Function name in a function call statement is considered as an operator

A function name in a functionical statement in a function call statement is considered an operator. Please remember whenever you are calling a function invoking a function the function name that in a function call statement is also considered as an operator.

(Refer Slide Time: 11:03)

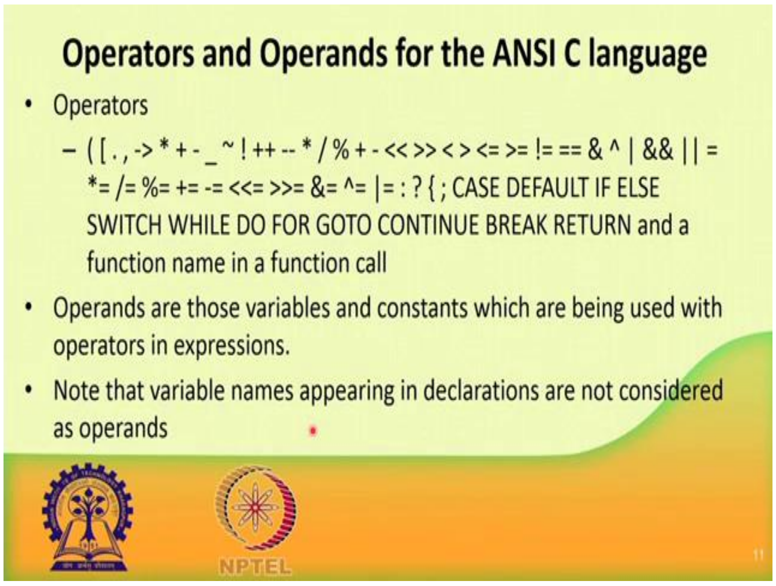

Guidelines for identifying operands cont...

- Subroutine declarations and variable declarations comprise the operands.
- The function name in a function definition is not counted as operator
- The arguments of the function call are considered as operands
- However, the parameter list of a function in the function declaration statement is not considered as operands.

The subroutine declarations and variable declarations they comprise the operands. So, now let us see about the we have seen some of the guidelines for the what operators. Now, let us see guidelines for identifying the operands. So, subroutine declarations, when you are declaring sub routines, so subroutine declarations and variable declarations, they comprise the operands. The function name in a function definition is not counted as an operator. Please remember that the function name while we are making a function definition, the function name that is not counted is not counted as an operator. So, similarly the arguments of the function call are considered as operands. So, whenever you are putting the arguments where in a function call the arguments of the function call they are considered as operands.

However, if the parameter list of a function in the function declaration statement is not considered as an operands. While you are declaring a function, if you are putting the parameters the parameter list that present in the function declaration, there that is not considered as an operand.

(Refer Slide Time: 12:17)



Operators and Operands for the ANSI C language

- Operators
  - ([. , -> * + - _ ~ ! ++ -- * / % + - << >> < > <= >= != == & ^ | && || = *= /= %= += -= <<= >>= &= ^= |= : ? { ; CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN and a function name in a function call
- Operands are those variables and constants which are being used with operators in expressions.
- Note that variable names appearing in declarations are not considered as operands

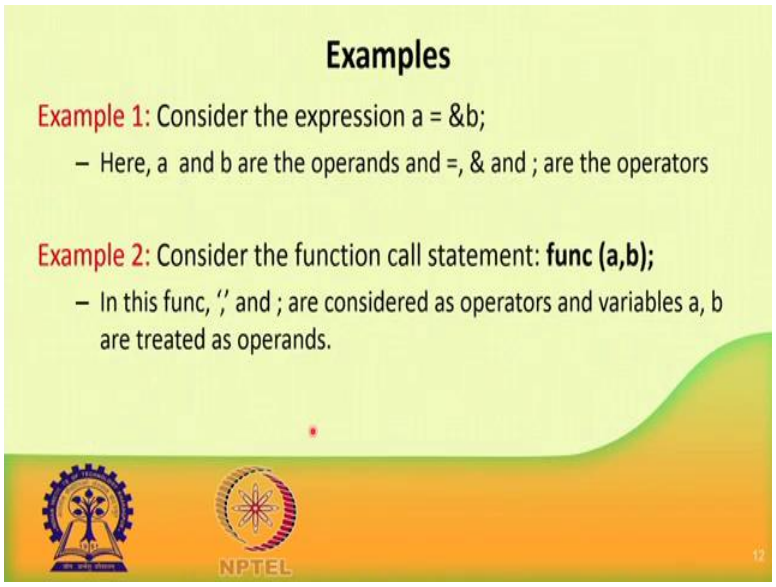
Let us see the common language, programming language C, language what to do the what a possible operators, so these are the possible operators like

> ([. , -> * + - _ ~ ! ++ -- * / % + - << >> < > <= >= != == & ^ | && || = *= /= %= += -= <<= >>= &= ^= |= : ? { ; CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN and a function name in a function call

these are all examples of operators valid operators in C as a as long as this Halstead's software science is concerned.

So, the operands are those variables and the constants which are being used in operators in expression. So, in an expression, ok, in an operator, the operators while you are putting in what in expression the operators while you are putting the expression. So, we have to consider the operands also. The operands are those variables lets define the operands operator since you all already have defined, let us define the operands, operands are those variables and the constants which are being used with the operators in the expression. So, along with the operators, so what variables and constants you are using in the expressions, they treat it as operands. Note that the variable names appearing in the declarations, they are not considered as operands. If you are using some variable names in the declarations, they are not considered as operands.

(Refer Slide Time: 13:46)



Examples

Example 1: Consider the expression a = &b;

- Here, a and b are the operands and =, & and ; are the operators

Example 2: Consider the function call statement: **func (a,b);**

- In this func, ',' and ; are considered as operators and variables a, b are treated as operands.

12

So, these are the guidelines we have given for for identifying the operators and the operands. Now, let us take this example. Consider that the expression I give a small expression is given

$$a = \&b;$$

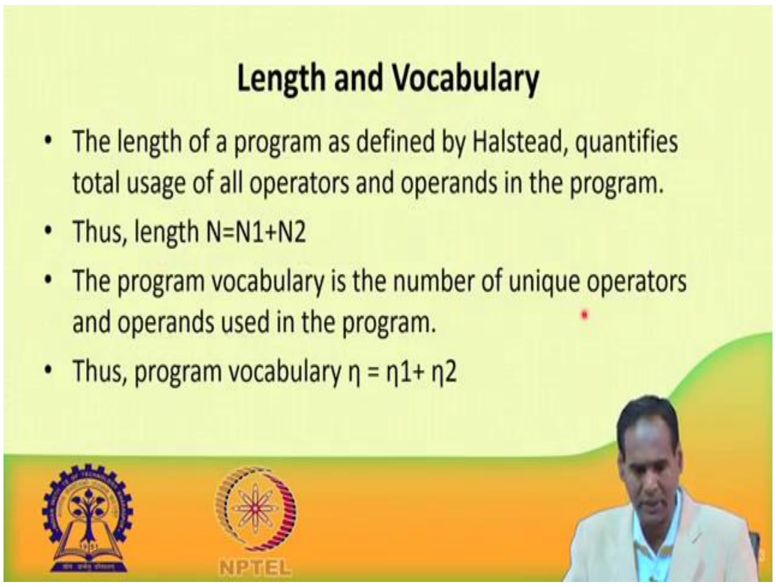
So, in this example, we can see that a and b are the operands, whereas equal to, ampersand sign and semicolon, these are the operators.

Similarly, if there is a function call statement

func (a,b);

then in this function, this function name func, then this comma in present in between a and b, and this semicolon operator, these are considered as the operators, whereas variables a and b they are treated as the operands. So, in this way given a program, you can easily identify the operators and operands this will help you for how to derive the expression for this volume, effort and the time etcetera.

(Refer Slide Time: 14:45)



Length and Vocabulary

- The length of a program as defined by Halstead, quantifies total usage of all operators and operands in the program.
- Thus, length N=N1+N2
- The program vocabulary is the number of unique operators and operands used in the program.
- Thus, program vocabulary $\eta = \eta 1 + \eta 2$

Now, let us define two other things important things called as the length and the vocabulary. The length of a program are defined by Halstead it quantifies the total usage of all operators and the operands in the program ok. The length of a program is defined as follows. The length of a program as defined by Halstead. What does it do; what does it do? It quantifies the total usage of all operators and operands in a program.

So, we have earlier taken N 1, N 2, and eta 1, eta 2, Please I am recalling again, whereas what is eta 1; eta 1 be the number of unique operators used in the program, eta 2 be the number of unique operands used in the program, and N 1 be the total number of operators, N 2 be the total this N 1 be the total number of operators used in the program, whereas, N 2 be the total number of operands used in the program.

So, with this we can easily and we have seen the guidelines how to select the operands, how to identify the operands and operators. We can see the length can be defined by using this N 1 and N 2. So, length is equal to

$$N=N1+N2$$

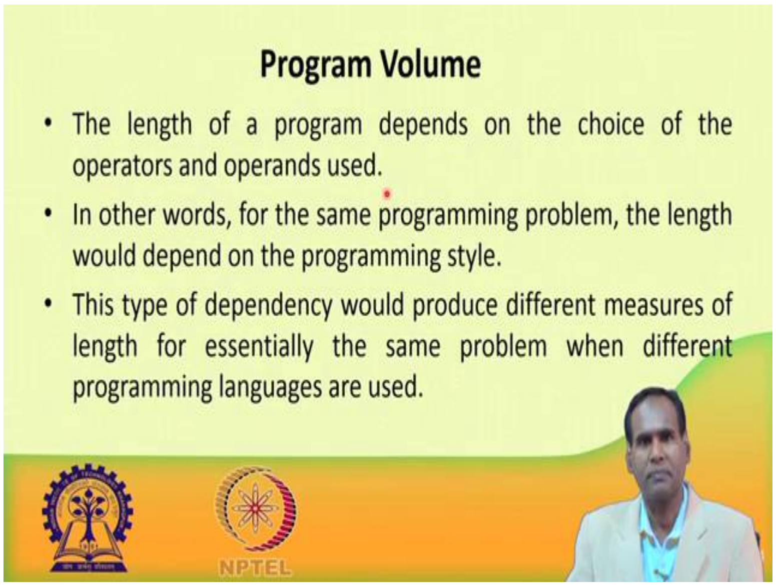
as per the definition because as according to Halstead the length of a program is defined as or it quantifies the total usage of all the operators and operands, simply add the operators and operands. So, this will give, so N is equal to N 1 plus N 2.

The program vocabulary it is defined as the number of unique operators and operands used in the program, not total operators and operands. It is defined as the number of unique operators and operands used in the program. Thus, program vocabulary eta is equal to

$$\eta = \eta1+ \eta2$$

that means, unique operators used in the program plus the number of unique operands used in the program. So, in this way, we can define we can estimate the length and vocabulary for a particular program by using Halstead's software science.

(Refer Slide Time: 16:53)



Program Volume

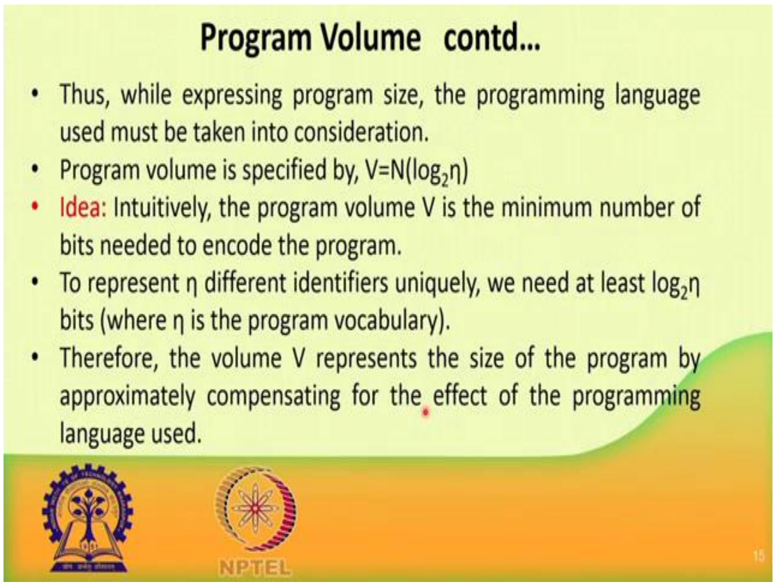
- The length of a program depends on the choice of the operators and operands used.
- In other words, for the same programming problem, the length would depend on the programming style.
- This type of dependency would produce different measures of length for essentially the same problem when different programming languages are used.

Next parameter is program volume. How we are defining program volume? The length of a program depends on what the choice of the operators and operands used ok. The length of a program it will depend on the choice of the operators and operands used in the program.

In other words, for the same programming problem, the length would depend on the programming style, because different programmers they will write down the programs using different styles. So, the same, for the same problem, for the same project, for the same programming problem, the length would depend on the programming style that the programmer has chosen. This type of dependency would produce different measures of length for essentially the same problem when different programming languages are used.

So, when different programming languages are used even if the problem is same to be that has to be coded, so that will produce different measures, different values for the length ok. This type of dependency would produce different measures of length for essentially the same problem when different programming languages are used by different programmers.

(Refer Slide Time: 18:06)



Program Volume contd...

- Thus, while expressing program size, the programming language used must be taken into consideration.
- Program volume is specified by, $V=N(\log_2\eta)$
- Idea: Intuitively, the program volume V is the minimum number of bits needed to encode the program.
- To represent η different identifiers uniquely, we need at least $\log_2\eta$ bits (where η is the program vocabulary).
- Therefore, the volume V represents the size of the program by approximately compensating for the effect of the programming language used.

15

Thus, while expressing the program size, the programming language used must be taken into consideration ok. Thus, while expressing the program size, what you have to do? The programming language that is used it must be taken into consideration. Program volume is usually specified by the term V which is equal to

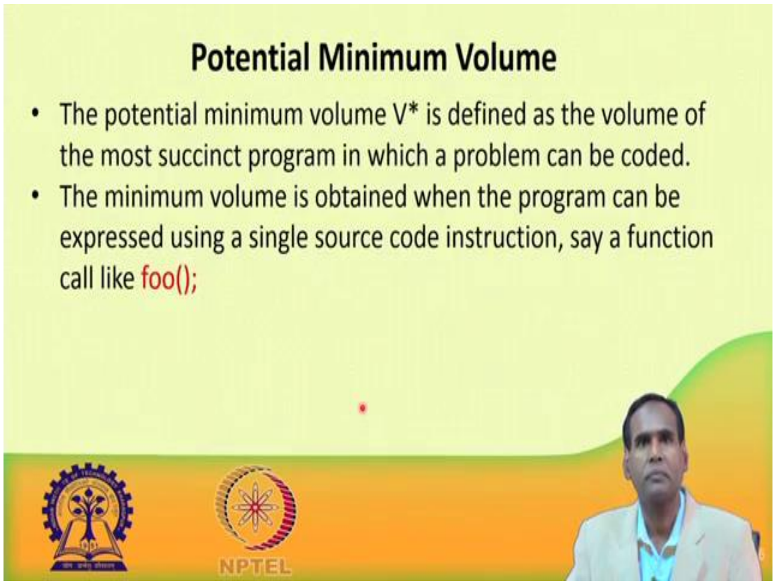
$$V=N(\log 2\eta)$$

N, you have already known. Total number of N is the what V is the program volume we are saying, and N just we have seen N is the N 1 plus N 2. Where eta is, that means, total number of operators and operands used. Where eta is equal to eta 1 plus eta 2 that is the total number the unique number of operators and operands used. So, program volume is specified by the term V which is equal to N into log 2 eta.

The idea behind is that intuitively the program volume V if the minimum number of bits required to encode the program. So, what is the minimum number of bits required to encode the program that is specified by V. So, in order to represent eta different identifiers uniquely, what we need? We need at least log 2 eta number of bits, where eta is the program vocabulary.

Therefore, the volume V, it represents what? The size of the program by approximately compensating for the effect of programming language used. So, if you are using different programming languages in order to compensate the effect of the programming language used, you can use what V which represents the size of the program. Therefore, the volume V, it represents this size of the program by approximately compensating for the effect of the different programming languages used.

(Refer Slide Time: 20:13)



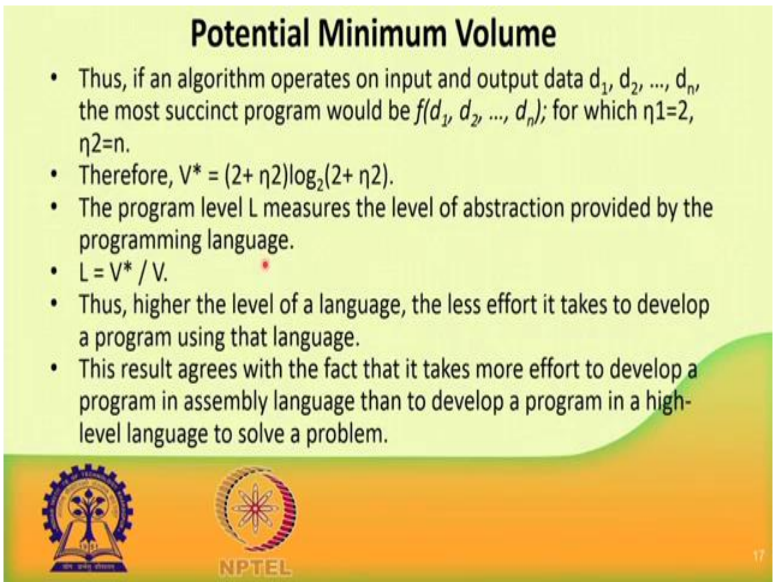
Potential Minimum Volume

- The potential minimum volume V* is defined as the volume of the most succinct program in which a problem can be coded.
- The minimum volume is obtained when the program can be expressed using a single source code instruction, say a function call like foo();

Next term that we have to discuss is program potential minimum volume. We have already seen simple volume. Now, let us see potential minimum volume. The potential minimum volume which is denoted as V star, it is defined as the volume of the most succinct program in which a program can be coded ok. So, the volume of the most succinct program in which a program can be coded is represented at V star or which is known as a potential minimum volume.

For example, let us see the minimum volume is obtained, when; the minimum volume is obtained, when the program can be expressed using a single source code instruction. Just like we say a function that they are small function foo, just a single code source code instruction, then we say that the minimum volume which obtained because the program can be expressed using just a single source code instruction, for example, a function call like foo(); It contains only one statement here ah

(Refer Slide Time: 21:15)



Potential Minimum Volume

- Thus, if an algorithm operates on input and output data $d_1, d_2, \ldots, d_n$, the most succinct program would be $f(d_1, d_2, \ldots, d_n)$; for which η1=2, η2=n.
- Therefore, V* = (2+ η2)$\log_2$(2+ η2).
- The program level L measures the level of abstraction provided by the programming language.
- L = V* / V.
- Thus, higher the level of a language, the less effort it takes to develop a program using that language.
- This result agrees with the fact that it takes more effort to develop a program in assembly language than to develop a program in a high-level language to solve a problem.

Thus, if an algorithm it operates on inputs and output data d 1, d 2 and d n ok. So, suppose there is an algorithm form if an algorithm operates on input and output data d 1, d 2, d n etcetera, the most succinct program would be how much f of d 1, d 2, d n for which eta 1 will be equal to 2 and eta 2 will be equal to n. So, eta at the minimum number of sorry the unique number of operators will be 2 and the unique number of operands will be n. Therefore, V star or the potential minimum volume will be equal to

$$V^* = (2+ \eta 2)\log 2(2+ \eta 2).$$

The program level, so now, we will define another term called as program level. The program level which is represented as L, it measures the level of abstraction which is provided by the programming language. So, a particular programming language what is the what level of abstraction it provides that is represented as program level L. L can be specified as

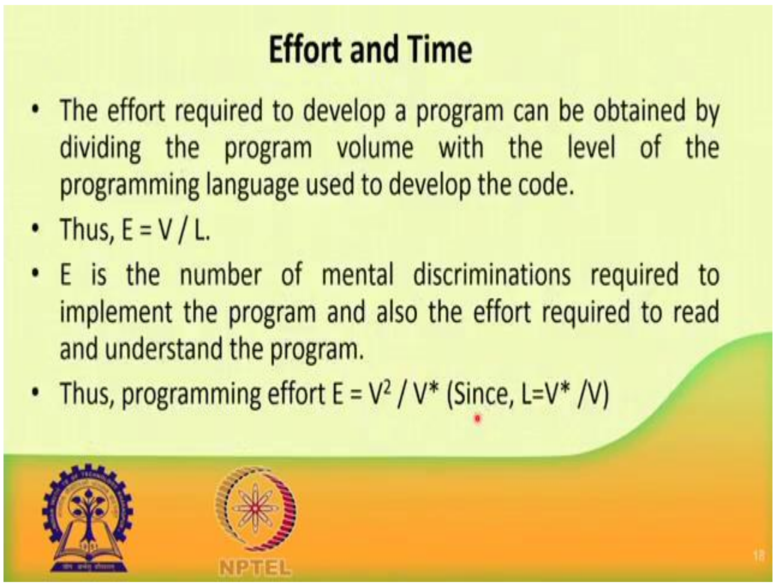
$$L = V^* / V.$$

where V star is the what potential minimum volume and be the volume. Thus, what will happen for the higher the level of a language the less effort it takes to develop a program using that language.

So, if the level of a what language is high, then it will take less effort to develop the program using that language. This result agrees with the fact that it makes more effort to develop a program in assembly language, than to develop a program in a high level language to solve a problem. We know that here what it takes more effort when where developing in the assembly language, because it is a low level language. Whereas, if we will take a what low level language which the high level language, then we normally put less effort.

So, it match with this thing that because L is equal to V star by V means it implied that higher the level of a language less will be the effort it requires to develop for that program using that language. And normally from our normal intuition, we know that if we want to write a program, then if will choose this assembly level language which is a low level language, it will take more effort than or in comparison to the thing that if you will choose a high level language such as a C or C plus to write down the same program.

(Refer Slide Time: 23:58)



Effort and Time

- The effort required to develop a program can be obtained by dividing the program volume with the level of the programming language used to develop the code.
- Thus, E = V / L.
- E is the number of mental discriminations required to implement the program and also the effort required to read and understand the program.
- Thus, programming effort $E = V^2 / V^*$ (Since, $L=V^* /V$)

Next we will define effort and time, what is exponentially required. The effort required to develop a program can be obtained by dividing the program volume with the level of the programming language is to develop the code. So, here already we have seen earlier using COCOMO technique etcetera how the effort can be estimated.

Here the effort which is required to develop a program can be estimated by dividing what? By dividing program volume with the level of the programming language which will be used to develop the code. Mathematical you can say that the effort will be equal to

$$E = V / L.$$

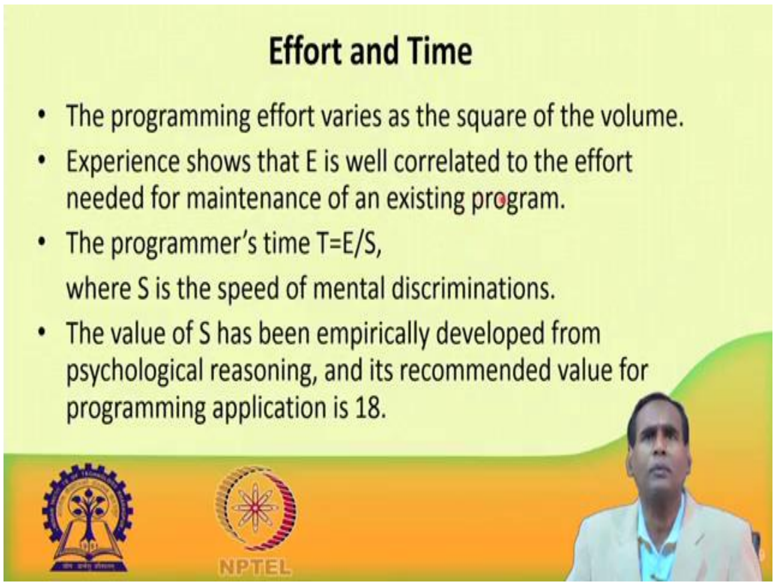
L represents the what is the level of the programming language.

Here, E is, so normally; so normally E the can be treated as the follows, normally E is the number of mental discriminations required to implement the program ok. So, normally E is the number of mental discriminations required to implement the program and also the effort required to read and understand the program. Thus, programming effort E can be written as how much will put the value of L, we know value of L is equal to V star by V.

So, putting the value of L here we get E is equal to how much putting the value of L is equal to V star by V. So, on simplification, it will give

$$E = V^2 / V^* \text{ (Since, } L=V^* /V)$$

(Refer Slide Time: 25:44)



Effort and Time

- The programming effort varies as the square of the volume.
- Experience shows that E is well correlated to the effort needed for maintenance of an existing program.
- The programmer's time T=E/S, where S is the speed of mental discriminations.
- The value of S has been empirically developed from psychological reasoning, and its recommended value for programming application is 18.

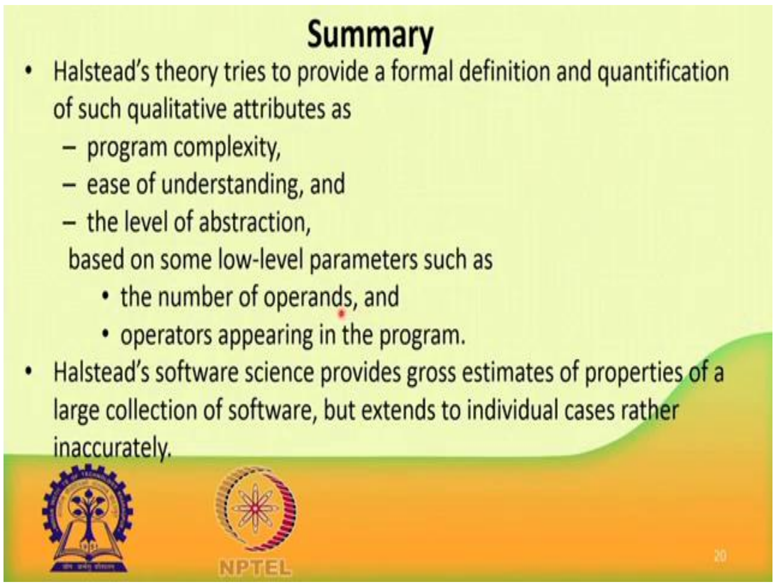
The programming effort varies as the square of the volume. You can see from this what equation the programming effort E it varies how; the programming effort E it varies as the square of the volume. Here you can see the programming effort E varies as a what a square of the volume V, the programming effort varies as the square of the volume.

Experiences shows that E is well correlated to the effort needed for maintenance of an existing program. So, Halstead has conducted so many so much of what experiments also other researchers has what conducted so many what experiments. And the experience shows that E is well correlated to the effort which is needed for maintenance of an existing program. Then another, what a term will define here that is the programmer's time.

Programmer's time is defined as $T=E/S$

where E is the effort and S is the speed of mental discriminations. The value of S has been empirically developed from psychological reasoning, and it is recommended value and this recommended value for programming application is 18. So, normally the value of S for programming applications is 18 that means, what the speed of mental discriminations is normally it should be set to 18. The value of S should be taken as 18 for programming applications.

(Refer Slide Time: 27:19)



Summary

- Halstead's theory tries to provide a formal definition and quantification of such qualitative attributes as
  - program complexity,
  - ease of understanding, and
  - the level of abstraction,

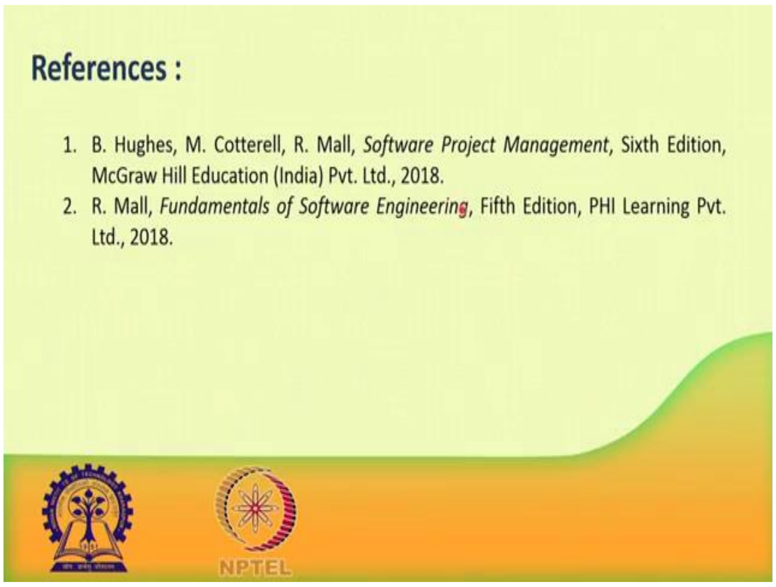
  based on some low-level parameters such as
    - the number of operands, and
    - operators appearing in the program.
- Halstead's software science provides gross estimates of properties of a large collection of software, but extends to individual cases rather inaccurately.

We have seen in this class the about Halstead software science which is an analytical estimation method. A Halstead's theory, it tries to provide you formal definition and quantification of such quantitative attributes such as program complexity, ease of understanding and the level of abstractions etcetera, based on some low-level parameters such as the number of operands, and the number of operators which are appearing in the program.

Halstead software science provides gross estimates of properties of a large collection of software, but extends to individual cases rather inaccurately. So, one thing we have known that Halstead software science can be used effectively for estimating what maintenance effort.

Halstead Software Science also provides gross estimates of a property of a large collection of software, but when extends to individual cases it, but extends to individual cases, rather inaccurately, for individual cases, it may give inaccurate estimations this is something about this Halstead software science which is an analytical estimation method.

(Refer Slide Time: 28:40)



References :

1. B. Hughes, M. Cotterell, R. Mall, *Software Project Management*, Sixth Edition, McGraw Hill Education (India) Pvt. Ltd., 2018.
2. R. Mall, *Fundamentals of Software Engineering*, Fifth Edition, PHI Learning Pvt. Ltd., 2018.

We have taken and the references from these two books ok.

Thank you very much.