This is not exactly a text, a midway between written text and speech, it is a written version of a lecture given to the Refresher Course for Teachers of Applied Psychology in Calcutta University. To be published in the collected papers of this Refresher Course that was held in University Staff Academic College, Rajabajar, September, 2005.

Tongue to Fingers: Colonizing IT in a Postcolonial World

dipankar das, dipankard@gmail.com

The first part of the title, 'Tongue to Fingers' is just a metaphoric way to mark out the drama of difference between the Command Prompt and the GUI (Graphical User Interface) Prompt, or, to say it just plainly, the difference between the Keyboard Way and the Mouse Way of working with a computer. But what is so dramatic, what is the big deal there, connecting it to the process of colonizing in a postcolonial world? We are coming to that, in a bit. First let us pave our ground, let us first come in terms with the terminologies: some piecemeal discussions on some important concepts from IT, the realm of Information Technology. Anyone even with an elementary familiarity with computer related things and events can directly start from section 4, the first three are for people from outside this realm, a description that exactly suits this audience.

Then we go into the second part of our discussion, from section 4, where we relate our findings from the event horizon in the IT field to our concept of a process of 'nameless colonialism' in a postcolonial world. Nameless in the sense that we are getting colonized without a specific colonizing Power. In the times prior to WW-II, the colonizer and the colonized all had very definite identities. We knew that the British are our lords, like the Algerians knew that their lords were the French, and the countries of Latin America knew the Spanish as their lords. We all were slaves of lords with definite addresses. But, this is a time when you cannot name the lord any more and still that does not hamper the process of our being and becoming and remaining slaves.

We proceed to show that the IT things are actually parts of a bigger project of colonizing the Third World without declaring it. Colonizing it by changing the mindset of the subjects in Third World, by distorting the Third World Psyche, always already inscribed with a lack, a 'mimicry of overdetermination'. And that too by changing the cultural ambiance in the world of computers, the sector that controls all the sectors in a modern day fully grown capitalism. A sector that attracts into it the most brilliant of all young minds. A sector that, after a long time, proclaims the supremacy of mind over matter in a technology driven world. The 'mouse culture', the 'user friendly computer' and the 'outsourcing' are all just bits and pieces of the bigger mural. The focus here is the 'colonial mind', the colonial mind in a 'postcolonial world', fed and fostered through the IT by the Monopoly MNC-s. A mindset that works on:

Highlighting the "user-friendly" graphics: the logo-literacy of the American kind

Taking away "Language": Language is for the elite, while the mass consumes pictures

Taking away "Language" from the realm of computer thought

The whole Third World rendered just a repository of cyber-age clerks

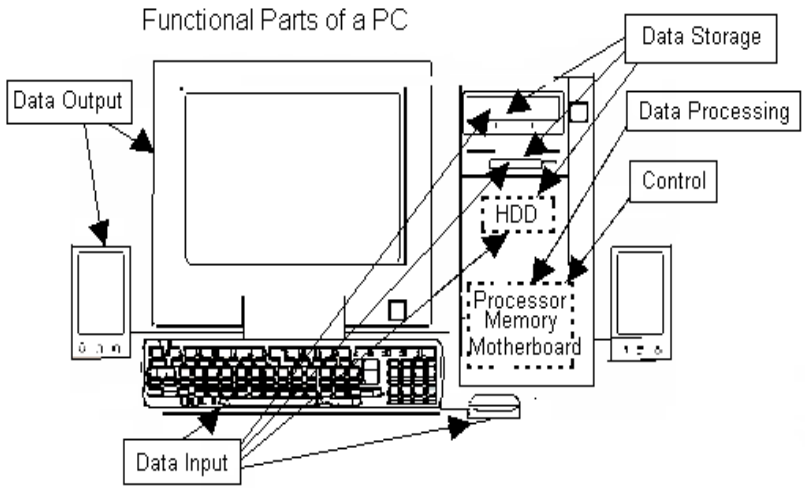
Increasing the World wide market of both Hardware and Software by all this.

1. The Computer and its Functional Parts

First, let us understand the working of a PC. For that let us inspect the familiar face of a PC in Exhibit 1. Here you can see the familiar parts. The console or terminal, where we see the output or the program running. This console and the sound-box have been tagged as 'Data Output'. The keyboard and the mouse have been tagged as 'Data Input'. The CD drive, Hard Disk drive and Floppy drive have been tagged as 'Data Storage'. And the electronic things within the tower, like

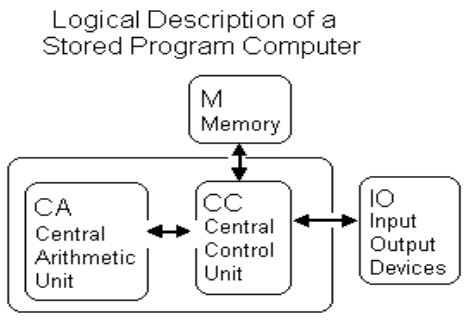
the chips and mother-board and all, these have been tagged as 'Data Processing' and 'Control'. We are coming to that: why they are tagged this way. But one thing keep in mind, not that these are the only possible parts or these are the only possible use of those parts. The printer can be used as an Output device too, or, even a CD or a Hard Disk can be used as an Output Device if you want, when you print your results, or, write it on a CD or save it on your Hard Disk. Now let us understand these tags.

Exhibit 1: The PC as we know it



All these tags with the component parts, like 'Data Output', 'Data Storage', 'Data Input', 'Processing', 'Control' Exhibit 1 actually relate the parts to Von Neumann Architecture, given in Exhibit 2. We are going to understand the basic functioning of a computer with this scheme done by Von Neumann. This scheme is called Von Neumann Architecture of Stored Program Computers. Let us get familiar with the scheme.

Exhibit 2: Von Neumann Scheme



What is there so important in this scheme? Say, we are working with a computer. What the machine is actually doing? And how the work is going on?

Whatever software or package or program we are working with, on whatever data, the work of the computer is essentially processing some data. Sometimes we are word-processing, that is using some program that allows us to format the text we are writing in our chosen ways, like, some special font or special alignment or special color and so on. Here the text is the primary data or Raw Data. And what we are going to do with formatting, like say, you selected some portion of the text, and went to the Font Selection and then made the font as Times and the font size as 12, as in this paragraph of this text, here the key-strokes or the clicks that you did to achieve this formatting – those actually sent the formatting data or Instruction Data. This is the essential run of things in every possible case. When you are playing some music, painting some picture, working on some database, putting some account to a spreadsheet, browsing the Net, sending some email or anything that you do with a computer.

You give the machine some data: data of essentially two kinds, the Raw Data and the Instruction Data, or, as a whole, Data. This action of giving the data to the machine is called Data Input. After the data goes into the machine, it processes the whole thing. Sometimes the circuits and the chips do some mathematical operation on it, sometimes some logical operation, and sometimes some operation that seems to us not directly from logic or mathematics, like, changing the color of some brush-stroke, but, to the computer it is nothing but some mathematics and some logic operations. Anyway that is not our point, the thing is, the computer now processes this data, logically or mathematically or both. But, here is another point. If the computer is processing the data, then, the data it has to process or the Raw Data and the data by which it processes or Instruction Data: both

of them must be held there for all the time the computer is processing it. If it cannot hold that much data for the required time, obviously it would not be able to do the processing works. This activity of holding the data is done in M or the Memory Unit in the scheme. And the processing part is done by the Central Arithmetic Unit or CA. And when all these works are over, the data comes out, say on the console, or the printer, or in some special cases, in some speaker or something. In the scheme the Data Input and the Data Output are put together in the block Input-Output Devices or IO. Actually it is written as I/O.

Has it occurred to you that one part of the scheme, the very central part, that is the Central Control Unit or CC we have not touched till now. What this thing is? This is better demonstrated than discussed. Say you are doing some very simple sum, 2 + 3 = ?, with your calculator. What you are doing? You are going to the Data Input unit, that is the keypad of the calculator and keying in the keys exactly in this sequence: '2', '+', '3', and '='. No other sequence will bring you the wanted result. And now the result is ready for you on the Output Device or the display unit or the screen of the calculator. You can remember this result, or, write it down somewhere if it is a series of calculations. This is Data Output. And see the data you keyed in were of two kinds. The data comprising '2' and '3' was raw data, and the other part comprising '+' and '=' are Instruction Data. Now think about this whole process of calculating with a calculator. All the bits of action, input of data in its proper sequence, writing out the output, and then again going into a new calculation – what/who is controlling this process? You. You are the one that is controlling it. Because the calculator is not a computer, and hence it does not have a CC. Till computers came, the CC remained in the head of a human being and his rough notes. For the first time in the history of machines man could vest this CC on to a Stored Program Computer. In a bit we will see how a stored program can contain the Raw Data, the Instruction Data and the algorithm or procedure of controlling it. And this is exactly the job that the computer does. And so comes the control block in the scheme, CC, that actually sits in the central position and connects with all the other blocks. This Scheme in Exhibit 2 was done by Von Neumann, the celebrated physicist-mathematician-computer-scientist. Why it is important? Because for the first time, in the history of machines, there was a machine with a difference, a machine that started to take away some burdens from deep inside our head. A machine that got vested with a part of us: the computers. And that can be demonstrated with this Exhibit.

2. The Role of the OS: Operating System

Exhibit 3: Different Layers of Components and Functions in a Computer

Layers from Hardware to Software				
6	Accounting ...	Railways ...	Internet ...	Layer of Application Software
5	Compiler	Editor	Command Interpreter	Central Layer of System Programs
4	Operating System			
3	Machine Language			Hardware Layer of Physical Components
2	Micro Architecture			
1	Silicon Metal Plastics			

Now that we have got the concept of CC and its working in a computer, let us proceed a bit towards the real working of a real computer. How this very elementary scheme of operations can stand for a complex thing like a modern PC going on doing all those things together? For that, we need just an ounce of discussion about the Operating System. What is an Operating System or OS? To speak the paparazzi way, it is the program that works as the mother of all programs in a computer. It takes all the trouble of remembering all the intricate hardware details and generates an easy-understandable virtual structure of the machine to allow us use it comfortably. In a very nice way this whole thing is dealt in 'Modern Operating System' by Andrew Tanenbaum. The structure of the table that we are going to use here is taken from that book

too. We will just touch it here: we do not need the details.
This table is pretty straight-forward. At the lowest level lies the layer of hardware. This hard world actually is divided into sub-layers of machine details and the most elementary form of communication: a pre-language that sends the commands and instructions into the direct hardware: the world of micro architecture and Machine Language residing within this planet of silicon chips and wires and metal junctions and all those. Actually this is the layer that we can almost totally forget while operating the computer thanks to the OS.

Above the lowest layer of Physical components, comes the layer of System Programs. This layer is again divided into two sub-layers. The lower sub-layer here is the OS. This actually creates a kind of blanket over the hardware details. And generates a virtual space, a space that is intelligible and easily understandable by human beings, that they can use it. One very small example we will cite here in moments. This OS sub-layer is the base on which stands the three major components of System Program: the core of a working computer. These three components are Compiler, Editor, and Command Interpreter. In the next section we will see how a piece of code is written and then compiled by a compiler to generate an executable program: a program that runs. And everything that runs in a computer is a program or combination of programs. Working with computer is nothing but running programs. To write this code we need an editor. Then to compile this code we need a compiler. And while doing all this to operate within the system we need a Command Interpreter. A Command Interpreter, usually called a shell, is the middleman between you and the operating system. An over-simplified biological analogy can be like this: a machine without an OS is nothing but a body without any life force, a dead piece of flesh, and, command interpreter is the language with which it communicates with the reality around it when it gets alive.

A lot of you, being familiar with MS-Windows and MS-Windows only, will not feel at home with the concept of a shell. It will seem to a lot of you: what is this shell, why would I need it. Whenever you do some job other than running a program, it is something connected with the system, like making a directory, copying a file and so on. And more often than not, to do this job, you open Windows Explorer and do it with your mouse, like selecting something, dragging something, or right-clicking on it, and then clicking something from the menu that has popped up. Even to run a program quite frequently you double-click on it with your mouse or something like this. But, the thing is that at every instance of your every action a very active shell is there, looking after your every request, ensuring that it gets executed. But, that is hidden from you. Why? That is the whole point of this effort. And how? We are coming to that very soon. Every time you do something with your mouse that is translated by shell into a command and the command is accordingly sent to the OS: that is the job of the shell, that is, the Command Interpreter. And when you are running some program, it is shell that is finding out the program and running it for you, after OS allows it do so. One very simple example of not getting allowed may be when you are wanting to change some file while it is being used by some other program. These three, Command Interpreter, Compiler and Editor come together to form this layer of System Programming, that is, how the system runs.

And now comes the third layer: Application Softwares. The things that concern an average user, like the word-processor, the database application, the browser, the music player, and so on. That an user must not look beyond these things, not just a consumer-user, but power-users too, is actually a part of the politics of colonizing the Information Technology in a postcolonial world: this is where we want to reach.

This multi-layered existence and presence of the OS, blanketing the raw intricate hardware details from our eyes, and representing the machine in an easily understandable virtual system is going to be clear to us by the example of a folder and a file in a particular partition. You are all quite familiar with these words. Your 40 GB hard-disk may have two partitions, in MS-Windows system that are called as ‘C:’ and ‘D:’, usually attached with a ‘:’ to denote its drive status, in MS-Windows environment a partition is usually is called a drive. Let us imagine, on the ‘C:’ drive of your computer there are two directories, ‘one’ and ‘two’. In Windows Explorer you can see the folders as two pictures. And, if you go to ‘MS-Dos Prompt’ from the ‘Start’ menu, you can access them as ‘C:\one’ and ‘C:\two’, and the files within ‘C:\one’ can be accessed similarly. Say, you want to

copy all the files of ‘C:\one’ directory to ‘C:\two’, you will give a command in the black console of the ‘MS-Dos Prompt’ like this:

```
copy C:\one\*.* C:\two\
```

Look at this command. There are more things in it than it seems. First, the command ‘copy’. It is actually a program that copies files. This program is put somewhere in the MS-Windows installation such that when you give this command the program can run. Then comes the two addresses, ‘C:\one*.*’ and and ‘C:\two\’. The program called ‘copy’ is made like that. The things in the first address are copied into the second address. And the ‘*.*’ there in the first address has the meaning that all files with a two segment name with a dot between them are copied. Say files like ‘autoexec.bat’, ‘config.sys’, ‘anyfile.doc’, ‘song.mp3’: all these names have that common structure ‘*.*’. And that is how all the files from the directory ‘C:\one’ are copied to ‘C:\two’. But, this is not as simple as it seems. When you are giving this command and thus executing the ‘copy’ program, the computer must know from before where the program is, how to run it, and obviously what is ‘C:’ or ‘one’ or ‘two’ and where they are. But this actually leads to more questions than it replies to. Who is there that is reading that symbolic representation ‘*.*’ and understanding it to be all files with that particular structure? And there is a bigger problem than this. The addresses are the problem. To understand that, we have to look at the Exhibit 4.

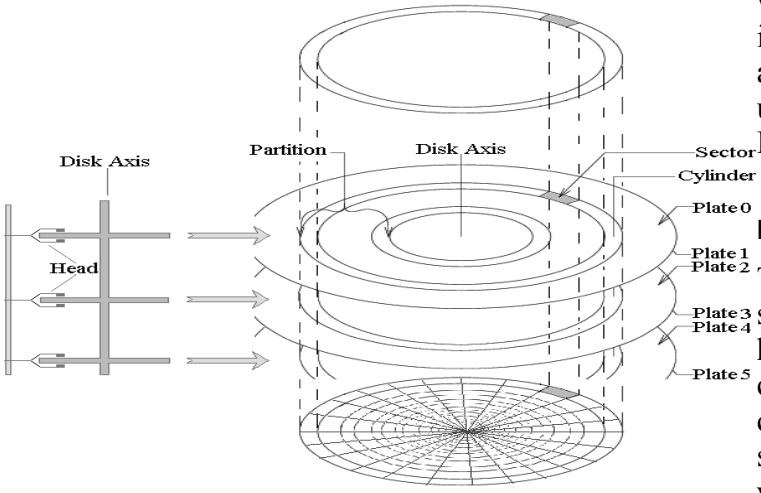


Exhibit 4: Within a Hard Disk

The Hard Disk is actually made of a series of platters. Each of these platters has two surfaces. Each surface is a collection of raw electronic space composed of millions and billions of small units of spaces, where you can write either 0 or 1. Anyway, that is not the point here. In the diagram the scheme of the action within a Hard-Disk is shown. Here we have shown three platters with six surfaces on which rotate six heads. These six heads rotate or move together. On every platter there is a series of concentric tracks. And so, when six platters move and on them move six heads, six corresponding tracks on six platters come together to form a cylinder, actually a virtual cylinder comprising of six tracks really existing on six real platters. Now, the partitions are made of cylinders. And so, it is a virtual construct. Because its unit cylinder is a virtual. Virtual in the sense that in real space they do not exist together but on six different real platters. Within these partitions, say ‘C:’ and ‘D:’ of your MS-Windows system reside the directories and files. So, in the real world of the real hard-Disk the residence of a directory or file is extremely complex, made of sectors within cylinders that actually is a made-up space of six platters. And the number of platters is much bigger than six. We just showed six of them in our diagram to get a feel of the real things. Now, just think, someone or something is always there, who or what is translating every human-understandable name of the partition or directory or file that you want to work with into some numbers, and tracking all those data from those sectors of the tracks of the platters of the Hard-Disk every time you want to do something. And this who or what is the Operating System. Not just the Hard-Disk, but every bit and piece of every hardware that you have in the machine is actually blanketed by the OS, the OS is remembering every details of it just to let to go free and work as you wish.

And then comes the shell. When you gave the ‘copy’ command, it was the duty of the shell to find out where is the program that you want to run, and then, again it was the duty of the shell to understand the symbol ‘*.*’ and send all the names of all the relevant files to the program, and thus, all the files of the directory ‘C:\one’ was copied into ‘C:\two’.

But both these OS and the shell are normally hidden from you, an average consumer-user, or any user in the MS-Windows environment for that matter. Why? Just for convenience? No, actually, a much bigger politics is there in this hiding. We are coming to that. But, to get into that we have still to understand one more thing: the process of compilation. Now before going into the next section let me remind you. Think about the whole command that you gave there on MS-Dos command prompt: '`copy C:\one\*.* C:\two`'. This command actually was translated by the shell into a much longer one, where the symbol '`*.*`' was replaced by the names of all the files of the directory. Then this long line of command that you are understanding as a series of characters was actually transformed into a series of numbers that finally reached the computing hardware of your machine. This series of numbers is the data. This contained both some 'instruction data' and some 'raw data'. Like, say, the name of the files were the raw data, and the instruction was to copy them from one location to another. Both these kinds of data was transformed into a series of numbers because all the alphabets and the characters that you are using to do it is only human-understandable. The machine does not understand it. So, something somewhere, most probably in the program called 'copy' is there that transformed this data into numbers and executed it through the computing devices. So, now, we need to understand the structure of a program and how it works with humans on one end and machines on the other. For this we need to understand a compiler.

A compiler compiles a 'code', written in some high-level language like Fortran, C, C++, Java, and so on. These languages are called high-level to denote that they are human-understandable. The machine language at which the real computation occurs is, in contrast, called the low-level language. A human programmer writes the code, where all the things are put which the programmer wants to be done. But, because the human programmer can read and write and understand it, it is high-level, and hence the machine does not understand it. So, this piece of code has to be translated into a machine understandable low-level language such that it can be executed by the machine.

Exhibit: 5: A C Code

```
#include<stdio.h>
int main(void)
{
      int x, y, sum;
      x=2;
      y=3;
      sum=x+y;
      printf("\n 2 + 3 = %d \n",sum);
      return 0;
}
```

This program does exactly that thing what we were doing with a calculator, that is, "2 + 3 = ?". It takes x (=2) and y (=3) and adds them and show us the brilliant discovery that "2 + 3 = 5". Now, without knowing even an iota of C, let us try to understand this brilliantly stupid piece of C-code.

The first line '`#<stdio.h>`' starts with a '`#`', in majority of programming-language convention this actually stands for a comment. This comment is not directly gives any instruction to be executed, but tells something about the code, either to some programmer who happens to read the code or the machine. In this case, this comment tells the compiler to include something called 'stdio.h' in the code. The compiler knows from where to get the file called 'stdio.h'. This 'stdio.h', like all other '*.h' files, is a header-file, as it is called. For all of us non-technical people here, a good comparison can be reference books kept in the library. And, actually, this is called library. This first line tells that some things will be used in this code, to understand and execute those things this particular reference file from the library must be included in this code. The header files in the library, like the reference books, are some collections of code or instruction that can be accessed by any code. Otherwise that commonly-usable piece of code will have to be written every time one writes the code.

The next line, '`int main (void)`' is a system statement of the C language, as this code is written in C. The C language allows codes only through functions, and there must be a main function in every code. In this case the line declares that the main function in this C code will be void in the sense that it will have no value of its own, it will do some work and come out void. And also tells that the function will handle only integer values in its default. Line 3 and 10 just opens

and closes this function with an opening brace '{' and closing brace '}'.

Within the function, the first line, that is the fourth line of the code, '`int x, y, sum;`' tells us that this function will work with three integer variables, '`x`', '`y`', and '`sum`'. Then, the fifth and sixth line tell us that the value of variable 'x' is '2' and that of variable 'y' is '3'. Line 7 is actually the computation part of the same problem that we were doing with the calculator. It adds value of variable 'x' and 'y' and makes the value of variable 'sum' equal to that.

Now comes line 8 that now prints the value of the variable 'sum' on your screen in a particular format. And see here we have used a C function '`printf`', or the print function. This is the function that actually displays the results on the screen. And the first line of the code where we included the reference book on standard input-output or std-io or 'stdio.h' was actually relevant for us in using this print function. That is the reference from where this code learns how to take in some input or how to produce some output, be it on the screen or somewhere else.

And the last line, 'return 0' tells us that the function that we used here in this code, the main function, is actually void in the sense that it returns no value but zero. Though it has, obviously, other connotations too. As we told, this is just an oversimplified rendering of the whole thing. We just wanted a very flimsy taste of the real things that happen within a computer. But, the bitter irony of the whole thing is that, even after all these toils, this code is something that can never run on a computer, for the very reason that it is written in some high-level human-understandable language, and that is why we could understand a lot of things about the code even without knowing anything about C, and exactly that is why the machine will not be able to make head or tail of it. Now let us consider Exhibit 6. What can you make of it? Obviously nothing. And now the whole thing has turned on its head. This is the computer's turn. It will see very transparently this same code of Exhibit 5 in this, where we see nothing but gibberish.

Exhibit 6: The Program that Actually runs


```
^A`^@^@^@\uffff\uffff^B^@^@^@^@^@^@^@"^@^@^@^A^@^K^A^@^@^A^@^@^@\uffff^
@^@^@^@^@^@^@^@^@^@^@^A^A^@^@^A^@^@=^@P^@5^@P^@ESC^@P^@^T^
@P^@^P^@T^@?^@T^@^R^@\uffff^B^N^@\uffff^B^@\uffff^BQ^@Y^@.^@Y^@$^@Y^@^V^
@Y^@^O^@Y^@^D^@Y^@\uffff^B^@^@^B^@^@^B^@^@^L^@^^@^Ca^@\uffff^Ba^@s^Ba^
@9^Ba^@^Aa^@^Aa^@d^Aa^@W^Aa^@&^Aa^@^F^Aa^@^@a^@\uffff^@a^@P^@a^@1^@a
^@^@a^@^R^@a^@\uffff^A\uffff^@\uffff^A\uffff^@^A\uffff^@w^A\uffff^@a^A\uffff^@X^A\uffff^@
B^A\uffff^@=^A\uffff^@^V^A\uffff^@^@^A\uffff^@^@\uffff^@^@\uffff^@\uffff^@\uffff^@\uffff^@\
uffff^@\uffff^@\uffff^@v^@\uffff^@i^@\uffff^@Z^@\uffff^@P^@\uffff^@8^@\uffff^@^W^@\uffff^
@^N^@\uffff^@^E^@\uffff^@^_^@\uffff^@+^A\uffff^@^G^A\uffff^@^@\uffff^@^@\uffff^@n^@\u
ffff^@a^@\uffff^@^Q^@\uffff^@^O^@^@^@^@c^@^@G^@^@^P^@^@>^@^@!^@^@^W^@^
@O^A^@^@^@N^@^@@^@^@$^@^@^L^@^@^O^@^@^Y^@^@4^@^@Q^@^@c^@^@^O
^@^C^A^Y^@^C^A<^@^C^A^N^@^G^A\uffff^@^G^A^@^G^A^@^G^A\uffff^C^@^@^N^@\uffff^
B^B^@\uffff^B^F^@\uffff^B^@\uffff^B^@^X^A^R^@^X^A2^@'^A^]^@'^A^U^@'^A^P^@'^A\uffff^
B*^Az^B*^A^B*^A^A*^A^A*^A\uffff^A*^A\uffff^A*^Ad^A*^A0^A*^A^^^A*^A^K^A*^A^@*^A^@*^A8
^@*^A&^@*^A^N^@*^A^P^@Y^AH^@Y^A^BY^A^CY^A\uffff^C\uffff^A\uffff^C\uffff^A\uffff^C\uffff
^A^Y^C\uffff^A^C^C\uffff^A^B\uffff^A\uffff^B\uffff^A\uffff^B\uffff^A\uffff^B\uffff^A^B\uffff^Ax^B\uffff
^Ab^B\uffff^AG^B\uffff^A9^B\uffff^A0^B\uffff^A^Z^B\uffff^A^U^B\uffff^A^A\uffff^A^A\uffff^A^A\uffff
^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffff^A\uffffL^A\uffff^A^@\uffff^
A\uffff^@\uffff^A\uffff^@\uffff^Ab^@\uffff^AH^@\uffff^AC^@\uffff^A/^@\uffff^A*^@\uffff^AESC^@
\uffff^A^N^@\uffff^A^B^D^@^@6^D^@^@^C^@^@^C^@^@^S^D@^@^D^E^@^@a^@^AQ^@
^AG^@^AD^@^A5^@^A+^@^A!^@^A^W^@^A^O^@^A^A^A\uffff^A^A\uffff^A^Au^A^Af^A^A\^A
^AR^A^AH^A^A5^A^A$^A^A^T^A^A^L^A^A^@^A^@^A^@^A^@^A\uffff^@^A\uffff^@^A\uffff^@^
A\uffff^@^Ao^@^Ad^@^A"^@^A^X^@^A^M^@^A^M^@^M^B0^@^M^B^^@^M^BO^@^M^B\uffff
^@^T^B\uffff^@^T^Bs^@^T^BS^@^T^BK^@^T^B(^@^T^B^U^@^T^B\uffff^@^_^B\uffff^@^_^B^
@^_^BA^@^_^B^_^@^_^B@^_^B^]^@*^B^@*^B^A,^B\uffff^A,^B\uffff^A,^B`^A,^B:^A,^B^A,^B^
@,^B\uffff^@,^BS^@,^BC^@,^B>^@,^B^N^@,^B
,^B^AH^B^AH^B^AH^B\uffff^AH^B\uffff^AH^B\uffff^AH^B\uffff^AH^B^AH^B[^AH^B*^AH^B^@H^
B^@H^B\uffff^@H^B\uffff^@H^B\uffff^@H^B{^@H^Bb^@H^B9^@H^B^_^@H^B^S^@H^B^@L^
@k^B^R^@k^B^@k^B2^@k^BQ^@k^BP^@w^BA^@w^B^@w^B"^@w^B^X^@w^B^M^@w^B^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
```


The C-code in Exhibit 5 does not run. We compile that code and get an executable file like this that runs, we call it a program. We actually gave the piece of code in Exhibit 5 to a compiler and got an executable file that runs, and gives us the result of the problem: '2+3=?'. In place of running the file I just copied a portion of the executable file here. This file was meant for running and hence it can communicate with the computer, but not with us, because this file was not meant for reading. This is what the compiler does. Transforms the human readable piece of code into a machine-understandable and machine-executable program.

And to write this piece of code we needed an Editor. See, this Editor allows us to enter some lines of text into a file, that is how we write a code, but, this Editor is so different from the Word-processors like MS-Word or Ooo-Writer or something like that, softwares that so many of us are so familiar with. Unlike those Word-processors, we can just enter or edit some text here in an Editor window,

not change their formats like bold/italic or font-size or font-color or so on. The text is kept in its most simple form. Because for the code nothing else is relevant.

So, now we return to Exhibit 3. Look at Row 5. Three components are there, ‘Compiler’, ‘Editor’ and ‘Command Interpreter’. We have got a glimpse of all three things here. These three together build up the layer of System Programming. And the surprising part of it is that this whole layer is normally hidden from a consumer user. By the term ‘normal user’ we presume an average computer user working in an MS-Windows environment. This is a very elaborate hiding. To get the full picture of it and to get a feel of the politics working behind it, we have to understand the mechanism of hiding. And for that we need to understand a GUI.

3. GUI or Graphical User Interface

Now, let us see, what would have happened if we did exactly the same work as represented by the command, ‘`copy C:\one\*.* C:\two\`’ without typing it, now by clicking a mouse. Most of you here, and most of the computer users anywhere do in fact go on copying files and doing all other things within their computer without ever knowing that a command like does even exist. What you usually do is opening the system in a browser window, like ‘Windows Explorer’ on many MSW systems, and then clicking open the folder ‘C:\one’ and selecting all the files there, and finally copying them from the right-click menu or dragging them to the ‘C:\two’ folder. Both these folders are just two pictures in the browser window, like everything else.

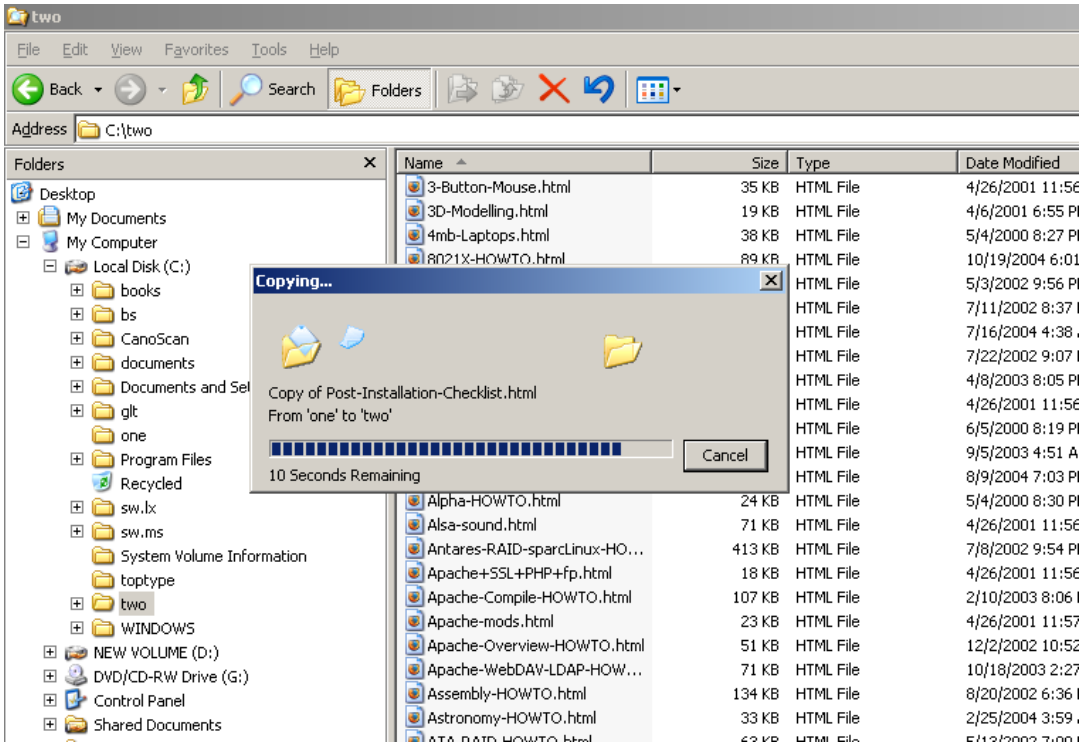


Exhibit 7: Copying all files or ‘*.*’ from ‘c:\one’ to ‘c:\two’ with a mouse

Check it from Exhibit 7. This is actually a still shot from the movie that started when you started copying the files. The partition ‘C:’ is a picture of a Hard-Disk on a palm, within that ‘one’ is a closed folder, to show that after copying the files from this folder you have now closed it, and ‘two’ is an open folder, to show that the files are now being pasted there. And the movie goes on, pages flying from one folder to another till the process of copying goes on. This way of working is called GUI or Graphical User Interface, because like words used in a language, here the unit of communication is a piece of graphics or picture. This GUI comes in contrast to the Command Prompt, where you have to type in a command, and after typing it when you press the ‘enter’ key, the command is executed. Like when you are copying those same files with the command.

This movie is actually a make believe one. When you are clicking active a movie like that, that is

copying some files, when you are clicking, the computer is actually translating your click into a command like that you used a bit back to copy those files like '`copy C:\one\*.* C:\two\`'. Then, that command gets executed, that execution is again re-translated into those moving pictures and held before you. This continuous translation and re-translation is the first layer of hiding. This hides the working shell. But the hiding does not end here.

The layer two in this elaborate process of hiding is hiding the commands. The commands and how they work. And everything associated with the real system working within a system: everything is hidden. And we shall readily see, there are more than one motives working behind it.

4. The language of mouse and picture against the Language

This GUI in place of Command-Prompt is a major shift in paradigm. GUI is a picture way of communication where the no of choices are always already pre-given and finite. You can only click at any of the given clicking point. And someone has already chosen for you what can come to your head. While, Command-Prompt works like language. All the commands are like words, that can be recombined and made into new commands. And all of them have quite a lot of options. These options are like special meanings attached with words. And this works through language too. We type some human-understandable and human-typable commands on the prompt, creating a direct interface between the man and the machine. But in a GUI it is always mediated and pre-configured. You can only react in one of the chosen ways. We are coming back to this point of paradigm shift in a bit. But before that we want to know more about the hiding.

With all these layers, what is hidden after all? It is the program-programmer-programmable-machine context. The biggest and widest Knowledge System ever created on the face of this planet. Because now everything is done with computers, this Knowledge System now intermingles and redefines every other existing Knowledge System that human civilization has created till now. When this man-machine interface gets hidden the very man-machine relationship actually undergoes a dichotomy.

As we have already commented, in the history of machines, computer was the first such machine that was vested with a part of man's mind: the reasoning part. And this part is once again taken away. And when it is taken away, a computer becomes exactly like an engine or a cassette player or even a tong. That never relieves us of any of our duties, be being remaining and never interrogating the status of a dumb instrument. So, with this interface taken away, computers only remain as typewriters, music players, TV-s, and calculators. Maybe a very versatile piece of instrument that can be used in so many jobs. But, nothing but a job-doer. The giant that is sleeping within the system, the thing that can directly interact with you, in an active way, by taking and executing your wishes exactly as you want them done, goes on sleeping. And it is never tapped on its shoulder, the way some software men like Richard Stallman, Linus Torvalds, Eric Raymonds and many many others, millions of GNU-Linuxers, wanted it to be tapped. They created full-fledged operating systems, compilers, software packages, that is millions of codes that is open to all, you can read and understand and change it at your will, and billions of pages of documentation that you can know and use and transform it as you want your system to be. A whole working system with every instrument that you may ever need to understand and participate directly in the interface between you and your machine is always already given in any GNU-Linux distribution like Mandrake, Redhat, Slackware, SuSE, and so on. All these are Free. Free in two senses of the term. Free to know and understand, what a commercial program from an MNC can never be, because if you understand it, you may not purchase it any more, you may create one of your own. They just give you the compiled programs, not the code that was used in the source to create the program by compiling it. The hiding that started from the command on the Command Prompt ended in hiding the code from the program. This is the first sense of the word 'free'. And the second sense is obviously that you do not have to pay anything to get it.

But, let us return to the computing way that is familiar to the most of you here in this hall,

computing in a system like MS-Windows. In a machine the usual way, you can get down only to the user level, that is the uppermost layer in our Exhibit 3. The layer that holds all the user programs. But, mind it, they are programs. So someone had to write the code and compile it, in most of the cases, some programming laborer rented in by MNC-s like Microsoft and so on did it. Did it for you. This 'for' part is very good, but it has an addendum: 'for you and without you'. You were not there. You just go on using it. You cannot know it, understand it, and change it.

Hiding of the typed-in commands was the first layer, then came the hiding of the shell and the operating system, and then came the hiding of everything related with programming or a direct interface between you and the machine you believe to be of your own. Apparently, this is a very good thing, in line with the concept of 'specialization'. But there is more to it than this. These MNC-s, naturally, cannot let you know or change the programs, because, naturally, it is copyrighted, but where from the right came? Where from the software programmers got the knowledge to do it? They got it the common part of all information: human civilization. Civilization is nothing but the totality of all these Knowledge Systems available to everyone. What these software engineers did. They just added a bit, a very minuscule bit in the true sense of the term, to the Knowledge System called civilization that is already existing. And by virtue of that microsopic something they want to take away the whole thing. And thus creates a system on your desktop that you have purchased but you cannot interact with it directly without the mediation of the MNC-rented software laborers. As the MNC model goes: the computer user remains a dumb instrument user, dumb in both the senses, dumb user of a dumb instrument, able to do only those very things that someone has pre-configured and pre-given. But why? Such a project is never without its politics. What is the politics of it? We will get to it in the last section of this lecture. Now let us examine this process of hiding in its actions and effects.

5. The Process of Paradigm Replacement

As we said, the Paradigm Replacement starts from the shift from language to pictures, but it does not end here. But, think of this starting point. How we got our alphabets? As it goes in Bangla, our "বর্ণমালা, দুঃখিনী বর্ণমালা" or, the sad alphabets, they were not delivered to us by some divine supreme, we reached it through thousands of years of human toil, physical and mental, from an animal like reality to pictures to pictograms to hieroglyphs to alphabets. And note here: exactly the reverse journey is getting traversed here. We are turning back from the world of alphabets to return into the world of pictures. And see, I do not know how far it is true, I have heard about a problem of alliteracy in a lot of Americans. This means knowing the alphabets once, but forgetting them in the true sense of the term, by sheer lack of use. I have heard that many Americans are there who have lost their reading ability that they had once in their schools because they never had to use it. They can recognize a coke by its logo, they absorb in the news from the TV programs, and like that. Even if it is not true, exactly a journey of that order is getting created here. I am coming back to this point readily. Before that let us get the other details and the cultural impact of this elaborate process of hiding.

Our sons and daughters, our little ones, from their very early childhood is getting to know computers these days. And in what form? In the shape of these pre-lingual pictograms. They are using the programs. And talk with any young one that you know, who is learning computers in the usual sense, you will discover that by 'learning' (s)he means 'learning how to run programs'. Yes, that is what is meant by learning computers in an everyday world. So, from the very start he is never trying to know how it is actually working. (S)he is becoming a computer consumer in place of a computer user.

And, as it happens, (s)he is growing up into an adult world, where some of them (read the brightest lot) are going to the world of computers. Here the same consumer-ambiance is continuing, and it is never interrogated. I know a young guy, quite a smart one, he did his degree on software from a very reputed institution and now is working in an equally reputed firm. And not only that, quite some years back, around 93-94 this is the guy that told me about GNU-Linux. And remember the

first Operating System by Torvalds and associates created by the software tools provided by Stallman and associates, was there on the net in the early nineties, and hence these were the budding years of GNU-Linux, and so, this guy was quite up and coming at that time. And now, what has happened to him, after working around six years in one of the most reputed computer firms in India? He cannot even install his own Linux system on his laptop on his own very confidently: and that is no big deal in any way, just tweaking a few configuration files here and there and understanding just a bit about partitions and file-systems and the concept of mounting. And as he himself has told me, he has actually become a quite-well-paid glorified data-entry clerk. My use of this word 'clerk' was not derogatory at all. Actually it is very history-conscious. I am coming to that. And these clerks on computer sometimes are making data-entry, sometimes are working with databases, sometimes just supervising a software system working on some field, a system that he cannot change or tweak or even understand, he just goes on supervising if it is working properly, and if it is not, he reports someone. Let us, as a whole call this community of computer-age clerks as cyber-clerks.

From the pictograms to consumer-users to cyber-clerks. That has become the development cycle of the brightest of our daughters and sons. The culture that nourishes them from their childhood, the economy that sustains them, all are actually breeding them like that. And Powers that be want it like that. From the very start, the shell, the Operating System, that is the kernel covering the whole hardware physical reality, was hidden from him. The whole environment of direct man-machine interface was hidden from him. And his occupational context does not prod him too to go into it now when his whole mentality has already been dwarfed. And this process of dwarfing is not anything new to us, citizens of a postcolonial country. Thomas B. Macaulay, the father of Westernized Modern Man in colonial India, asserted his viewpoints about a British colony, India like this: "... a single shelf of a good European library was worth the whole native literature of India and Arabia" (Macaulay, Thomas B. "Minute on Indian Education").

So, Macaulay went on making us worthwhile. Planned a system of education that will breed clerks. A lot of Indian clerks that are local and indigenous, thus hiking their efficiency in knowing understanding and being used as a ploy in the process of colonizing and oppressing and sapping out the Indian Economy. The British needed an efficiently working colony and hence they started modernizing us, better, some chosen few of us, the brightest lot of colonial India. The filtered these through the process of Western Mode of Education, Western theories and science, and bred some very good clerks, that can work efficiently in the Indian context. They understand the language of the British, know how the British system works, and, they are Indian. Importing British citizens are extremely cost-inefficient, and more than that, culture-inefficient.

Exactly the same way of colonial clerks, these postcolonial cyber-clerks are being bred. From the very start they are getting immersed in the logo-ambiance of the GUI environment, where you just search for the options that are already given there, learn the various options that are glorified as a 'computer course', and in the Examinations they are judged on the basis of their ability of memorizing the different pre-given options in the pre-compiled software packages. They cannot even opt for anything else. In the institutions and universities of the postcolonized India, the authorities, even without the slightest of shame, set the syllabus and ask questions on MS-Dos, MS-Windows, MS-Office. Let alone the question of working as an agent of some private capital, these all are something that no one, remember: NO ONE, outside the coterie of MS-rented software engineers can ever know fully well, because the codes that were compiled into these binaries, are copyrighted and a guarded secret.

Not that anyone expects every student of computer to become a successful code-writer. But, the point is that the very dwarfing goes on from the very beginning and that serves exactly the same purpose as the colonial project of generating the clerks did. Appropriating the Indian economy as a resource, not just in India, but in every country of the third world: the postcolonial world: all the countries that were colonies in their past. The same continuity is running here. This large scale production of cyber-clerks actually serves the postcolonial project on both the fronts of hardware and software. The software thing is to change and dwarf his mind, and the hardware project is

creating an endless demand of state-of-the art technology. While the Command Prompt can very efficiently work on even very old architectures, the GUI, with all its increasingly more intricate composition of an illusion of graphics interface, always suffers from lack of resource. In fact so many trillions of sticks of RAM-chips are just wasted on more colorful illusions which could be put to so many productive things. And if this is inefficient in terms of a Male White Rich economy like America, it is more than a crime for a Female Brown Poor economy like us. It is callousness. The movie playing in Exhibit 7 has a very high price tag indeed.

6. The 'nameless colonialism'

In our book, 'margin of margin: Profile of an Unrepentant Postcolonial Collaborator', we said that, postcolony has an unbroken continuity from the period of colonial history. In a lot of postcolonial texts the postcolonial is celebrated as an age of equality while the colonial stands for the oppression on the colonized by the colonizer. We said there, this very position has a politics of its own: why they say like that. No one actually is that stupid not to check it by asking any guy on the street, where would you prefer to be, here in India, or, there in America. So, this is just another name of being deliberately stupid. But, we dealt with those theories to quite a length there in this book. It is not the field to do it once again.

Here, we just want to say, the same kind of inequality is working here too. In the colonial days it was just a black and white equation. British are the lords, we are the slaves. But now, the power-game of postcolony has become nameless. You cannot exactly name any country like that who is colonizing us. But, the relation of inequality and power game is continuing still. And this manufacturing of cyber-clerks is just another ploy in this game. No country like that plays the game anymore, but the MNC-s do it for them.

We already mentioned this ploy works on two fronts for the MNC-s. Creates a continuing pressure in the market for more and more up-to-date hardware, that goes exactly in line with the explosion of consumer market. Televisions bigger and bigger, flat and flatter, and exactly the same way, 386 to 486 to 5 or 6 and so on. Hardly ever a situation can arise on a personal machine that demands computing power like that, except in very large scale compilation, and that is not a very common thing on a personal machine. So much resource are just lying in waste on so many desktops these days, very high power chips with virtually nothing to do. If that resource could be tapped in some other way, it could bring computing in reach for so many of those children of the lesser gods.

And, the most important point is the mentality of the postcolonial people. In a colonial world the biggest enemy resides within their own hearts. When the colonial subject looks to the mirror (s)he hates it, the mirrors tell it so surely it is not an European countenance. The subject then hates the subject's own self. (S)he wants to be a *Sahib* that (s)he cannot be. And this desire to become a *Sahib* makes the subject more un-*Sahib*, because a *Sahib* never desires to be a *Sahib*. The *Sahib* is automatically that. And because this colonial subject being a conscious subject knows of this anomaly. And this kills the self-esteem of the subject. From the very start he is constantly in war with the most intimate enemy: that is the self. The colonial subject endlessly and without a respite suffers from a lack of confidence. He is not at peace with his own self.

Just see, the same kind of mechanism is working here. The postcolonial subject, the brightest of them, the would-be-cyber-clerks, are from the very start thrown into an ambiance where he never wanted and adventured to reach out towards the machine and strive to come in terms with it, he starts using and continues to use a pre-configured and pre-compiled machine. So the prerogative of Knowledge remains a *Sahib* one still, exactly as Macaulay said. The libraries of the West remain there, we just get trained to become job-doers. Because the West needs low cost labor. Our people, our laborers have acquired the ability to go on working neatly with less meat and less liquor and less other things than the privileged Western ones. This is one advantage that should accrue to us. Even that is taken away by the West. We become the proud producer of a nation of cyber-clerks. Just think, what is 'out-sourcing' after all? It is cyber world system of putting out labor.

Not that there is no way out. GNU-Linux is there. That gives a total fully armed laboratory to go on experimenting and working, and thus knowing what a computer actually is. But so few takers remain there. Because, as we said, the enemy resides within. It is a culture of dwarfs that is deliberately generated. A culture that is pursued by the parents: that is us, dwarfing our own children. The process of dwarfing starts with the replacement of language on the Command Prompt by a picture and a mouse-click. It goes on. Takes away the shell, the Operating System. And then takes away all the programming languages too. Just job doing remains. It serves the postcolonial project.