reverse engineering Attempting to understand how a pre-existing device works on the basis of how it behaves.

Marr's levels of explanation and cognitive psychology

How, therefore, can such information processing devices be best understood? To answer this, we must now turn to a framework for thinking that was provided by Marr (1982). According to him it is possible to discuss three levels of description of any kind of information processing device, namely:

1. The level of the computational theory.
2. The level of the representation and the algorithm.
3. The level of the hardware implementation.

The level of the computational theory

At the **computational theory level**, concern is with what the device does and why it does it. It is at this level that the 'logic of the strategy' (Marr, 1982, p. 28) is spelt out. Consider the following example of an electronic calculator and the question of how it carries out arithmetic. Analysis at the level of the computational theory would address the fact that the calculator carries out various arithmetic operations (the 'what it does') and the fact that it uses a particular method for carrying these out (the 'why it does what it does'). For instance, an early Hewlett Packard calculator (the HP35) used a method based upon something known as Reverse Polish. So expressions such as:

$(1 + 2) \times 3$

were entered as

$1\ 2\ 3 \times +$

which accords with something known as postfix notation (or Reverse Polish – and because of this, and perhaps the price, it is no wonder this model quickly died out). The computational theory would be therefore be concerned with issues like why Reverse Polish was used and what the principles of Reverse Polish are.

The level of the representation and the algorithm

At the **representation and the algorithm level** much more detailed questions are asked about the nature of the calculator's operating system and the manner in which numbers and arithmetic processes are embodied in the device. In this respect, we are interested in how information is stored (i.e., represented) within the device and also how various arithmetic operations are instantiated. We will discuss in much more detail the notion of representation as we proceed through this book (see Chapter 7, for instance). For now, though, we will stick with a simple definition and assert that for any information processing device, information from the outside world is represented internally within the device. So when the number 2 is entered into the calculator this is represented via some form of electronic code. This form of **internal representation** stands for the number 2. By analogy, where the mind is concerned such internal (mental) states stand for (i.e., represent) actual states in the real world.

What then happens to such representations is a matter for the **algorithm**, or more particularly, the set of operations that are carried out on these representations. In computer science the term 'algorithm' is mainly used interchangeably with the phrase 'computer program', but we may take a more general reading and define it as a procedure that, when correctly applied, ensures the correct outcome. If you entered a '+' sign into the calculator and it is working properly then it should invoke its addition algorithm. The addition algorithm comprises the sequence of operations that determine that two numbers are added together. So understanding the nature of the calculator depends on trying to specify the nature of internal representations and the associated internal algorithm. In terms of understanding human cognition, and by analogy, we need to consider both mental representations and mental processes. In this respect our functional account should not only provide a flow chart that maps out the relations between component processes, but also some description of the sorts of internal representations that are also implicated. A much more thorough exploration of these ideas is contained in the next chapter, but in summary, at the level of the representation and the algorithm we are committed to being precise about (i) how states of the world are represented by the device, and (ii) what the contingent internal processes are.

The level of the hardware

Finally there is the **hardware implementation level** and, as we have already noted, flow charts such as that shown in Figure 1.6 are of little use. Concerns at this level are with how the designated representations and processes are implemented physically. What physical components do we need to build the device? As has been discussed, the one purpose of a functional description is to avoid any commitment to physical instantiation, but of course to attain a full understanding of any physical device details at all three levels of explanation will need to be addressed.

Pinpoint question 1.9

According to Marr (1982), what are the three levels of description for any device?

Levels of explanation and information processing systems

Let us examine the notion of levels of explanation in more detail and apply this in an attempt to understand the operation of a programmed computer: the paradigm case of an information processing system. One way of conceptualising how a computer operates is in terms of the following levels:

1. The intended program.
2. The actual computer program.
3. Translation.
4. Machine instructions.
5. Transistors.

The level of the intended program – Level 1 – is where discussion of what it is that the program is designed to do takes place. Using Marr's framework, this is couched at the computational level and may be fleshed out by stating what we want the program to do – we need to specify the goals and objectives behind the program. For instance, we want the program to produce a bank statement that itemises all in-goings and out-goings on an account in a month. At this point there is no need to make any reference to any specific kind of computer or hardware of any kind. At the next level (i.e., at Level 2) the necessary step is to commit ideas to 'paper' by writing a computer program in a particular language, e.g., BASIC, PASCAL, FORTRAN, etc. In completing these two stages both the level of the computational theory and the level of the representation and algorithm have been captured. Importantly, both of these levels have been addressed without any concern whatsoever about the nature of the computer that the program will run on.

For computer languages to be at all useful, however, there needs to be a means of converting (i.e., translating) an actual program into a form that can be run on a particular computer. Stated thus, concerns about hardware are paramount. In some programming languages the translation stage may involve two steps: (i) converting the computer program into an intermediate code known as assembly language, and then (ii) converting assembly language into machine instructions. It will be something of a relief to learn that we need not concern ourselves with the detailed nature of assembly language.

The critical point is that the stage of translation takes a representation of the program and converts it into something known as binary code – a series of 0s and 1s where each 0 and 1 is known as a bit. For instance, the command PRINT might be translated into 0101000010101010. The 0s and 1s are vital because they correspond to the respective OFF/ON states of the critical electronic components of your computer – here referred to as transistors. Think of these as being akin to simple switches in an electric circuit: if the switch is ON, electricity flows this way round the circuit; if the switch is OFF, the circuit is closed. So there is a fundamental level at which properties of the program correspond exactly with physical states of the machine. The machine code corresponds to the physical state of the computer's 'switches' and these cause the computer to behave in particular ways. Any change in state of the switches results in the computer doing something else.

This example provides a concrete illustration of how Marr's levels of analysis may be useful in attempting to explain the operation of a computer. The central idea is that the computer can be described at a number of different levels, and that there is no sense in which a description at any particular level is more correct or valid than a description at any other level. The different descriptions serve different purposes. If the questions are about what the program is supposed to do, then these concern the level of the computational theory. If the questions concern the design of the program, then these will be answered most appropriately at the level of the representation and the algorithm. If the questions concern whether the program uses an 8-bit or 16-bit representation of numbers (don't worry, your computer science friends will be only too delighted to tell you what these are), then the answers will be at the level of the machine code and therefore at the level of hardware implementation.