## 1.6 AN ILLUSTRATION

A company operates boats for hire on a pleasure lake. The customers come to the boathouse to ask the boatman whether he has a boat available for a session of sailing. If so, he makes the boat ready, helps the customer aboard, and keys a message into an online terminal. The message contains an identifier of the session and a code *S* to indicate that the session is starting; the terminal automatically adds the current clock time to the message. When the session is finished, the customer returns the boat, the boatman calculates the session time and receives payment from the customer, and keys another message into the terminal. This message again contains the session identifier and the clock time, and the code *E* to indicate that the session has ended.

(We may note that the management of the company has already chosen and purchased the terminal hardware, although the system specification is not yet written. This conforms to a practice widely established among some organizations.)

The manager of the operation wants some management information. He wants a report on each day's activity, in the form:

*NUMBER OF SESSIONS = nnn*

*AVRGE SESSION TIME = mmm*

where the average session time is, of course, the total of all session times divided by the number of sessions.

The system developer sees immediately that the required function has two parts: compute the information from the available input stream, and print it. Computing the information consists of counting the sessions, totalling the time, and calculating the average. Counting the sessions is easy: the number of sessions is the same as the number of S messages. Totalling the time is not much harder:

*totaltime = (endtime of session 1 - starttime of session 1)+*
*(endtime of session 2 - starttime of session 2) + ...*

or, more conveniently,

*totaltime = (endtime of session 1 + endtime of session 2 + endtime of session 3 + ...*
*- starttime of session 1 - starttime of session 2 - starttime of session 3 -...)*

The resulting solution, functionally designed and implemented in structured code, looks like this:

```
begin open message stream;
      get message;
      number := 0; totaltime := 0;
      do while not end-of-stream
         if code = 'S'
         then number := number + 1;
              totaltime := totaltime - starttime;
         else totaltime := totaltime + endtime;
         endif
         get message;
      enddo
      print 'NUMBER OF SESSIONS = ', number;
      if number ¹ 0
      then print 'AVRGE SESSION TIME = ',
                 totaltime / number;
      endif
      close message stream;
 end
```

The system goes into production, and everyone is happy. The solution provides exactly the function that is required. But soon the manager comes to the developer with a request for a `functional enhancement'. He wants the report to contain a third line:

*LONGEST SESSION TIME = ppp*

Careful examination of the existing system convinces the developer that the requested enhancement cannot be made except by throwing away what has already been done and building a completely new system. The reader should also examine the system as shown above, and convince himself that this is indeed so.

Fortunately, the developer persuades the manager that there are technical reasons, to do with the machine’s operating system, why the requested change cannot be made: the manager is dazzled by the technical brilliance of the developer’s explanations, and agrees that the change was not crucially important anyway. (Notice that no costs are debited to the system maintenance account: calculations of maintenance costs often omit the cost of not doing maintenance.)

Less fortunately, the manager is back soon with another request. This time he is sure the change will be easy, because all he wants is two reports, each one exactly like the report he is already receiving. One report should deal with the sessions starting before noon, and the other with the remaining sessions of the day. Once again the developer (and, I hope, the reader!) sees that the change cannot be made to the existing system. This time he devises an excuse based on the recent introduction of the fixed-block disks; the manager is somewhat less dazzled by the developer’s technical brilliance than he had been before, and is definitely dissatisfied with the service he is getting. Soon, he returns with yet another request. This time the change is vitally important: the telecommunication line is defective, and has been dropping some of the messages. As a result, there are some sessions for which only one message, either the *S* or the *E* message, is received. The system must be changed to exclude these sessions from the calculation of the report.

Unfortunately, this change is no easier than the first two: it simply cannot be made without abandoning the system and creating a new one. Neither the time nor the budget is available, and there is no way out. The developer’s explanations (something to do with the new compiler) fall on deaf ears, and he is soon updating his resume and seeking another job.

This text is Section 1.6 from:
Michael Jackson; *System Development*; Prentice-Hall International, 1983