The last decade has witnessed a considerable interest in the psychological aspects of programming. Section 2 presents an overview of research that advances cognitive models of how programmers construct software. The survey, which is selective rather than comprehensive, presents three cognitive models of designer behaviour as well as a brief exposition of closely related work. The taxonomy presented is based on the specific software development task (e.g. method finding, coding) under which the reasoning process is carried out by selecting a focal idea (i.e. a goal) and expanding this into a plan and then testing to see whether the plan satisfies the goal. He characterises the activity as being performed through evolution rather than design in that "the program is grown from seeds rather than designed". After many such plans have been developed the overall program construction proceeds by the retrieval and tailoring of such plans, if they exist for the particular problem. Otherwise a focal idea is selected and expanded as before.

Abstract

Recent empirical work carried out in the area of program design has advanced a number of alternative models of programmer behaviour. This suggests the need for a comparative study to investigate whether these models provide complementary or contrasting explanations. This paper briefly reviews three current cognitive models and reports on the findings of an experimental study involving the collection and analysis of video protocols of five experienced programmers. The discussion reveals that, although there is a significant degree of similarity in the explanations offered by the three models, they focus attention on different aspects of the program design task. Furthermore, the protocols gathered provide supportive empirical evidence for these models. It is also argued that the results of these works need to be consolidated into an explanatory framework. Moreover, the orientation of this framework needs to be predictive rather than descriptive so that the necessary guidelines for software development can be made.

1. Introduction

The last decade has witnessed a considerable interest in cognitive aspects of programming. Section 2 presents an overview of research that advances cognitive models of how programmers construct software. The survey, which is selective rather than comprehensive, presents three cognitive models of designer behaviour as well as a brief exposition of closely related work. The taxonomy presented is based on the specific software development task (e.g. method finding, coding etc.) on which the investigations focused. The specific of the observational study and the discussion of identified characteristics of behaviour with example protocols are respectively presented in sections 3 and 4. The results of the observational study are used as a basis for a comparative discussion which is presented in section 5. The conclusion strongly argues the need to consolidate much of the work currently being undertaken so that the empirical study of programming can move from the descriptive to the predictive.

2. Models of designer behaviour

The first model to be considered is primarily based on the works of Rist (1986a, 1986b, 1986c) and the works of Soloway et al. (Johnson and Soloway 1985, Soloway and Ehrlich 1984). Their works make extensive use of Schank's (1982) schema in its characterization and focuses attention on novices constructing/translation algorithms for relatively simple programming problems. In relation to the type of programming knowledge Soloway and Ehrlich suggest that programmers possess and employ programming plans. Plans are generic program fragments that represent stereotypic action sequences in programming. An example of such a plan is a "running-total-variable plan" which has the following framework:

\[
\text{sum} = 0; \\
\text{READ}(\text{new}); \\
\text{WHILE} \\
\text{sum} = \text{sum} + \text{new};
\]

Rist has investigated how these programming plans are developed in order to explain program generation at different levels of expertise. He has extended these ideas to proposed a "generative model of program construction" that views initial problem solving as corresponding to problem decomposition and solution composition. The decomposition process involves programmers attempting to satisfy achievable goals by the retrieval and/or generation of non-programming plans. If the required non-programming plans are found this is then followed by the retrieval of corresponding program pieces (roles, constructs, or schemas). Therefore program construction is a repeated process of retrieving and generating a plan and its associated code for each new sub-goal. In this process the new sub-goals may be created and coded in the attempt to expand particular part of the program and the need for different constructs, schemas, plans, or even goals, may be noticed by evaluating the current situation, and the generation cycle may be resumed.

His empirical observations with novices suggests that the construction/development proceeds by "focussing attention" on achieving the current goal to the exclusion of issues outside this context. The reasoning process is, therefore, not "top-down" but is carried out by selecting a focal idea (i.e. a goal) and expanding this into a plan and then testing to see whether the plan satisfies the goal. He characterises the activity as being performed through evolution rather than design in that "the program is grown from seeds rather than designed". After many such plans have been developed the overall program construction proceeds by the retrieval and tailoring of such plans, if they exist for the particular problem. Otherwise a focal idea is selected and expanded as before.
The second model to be considered is based on Siddiqi's (1984, 1985) research study involving six hypothesis testing experiments. The model views the problem solver as a set of knowledge structures relevant to program design, memory for storing and processing information and a facility for planning. The former is a complex multi-level body of concepts and techniques, often referred to as semantic knowledge, and includes general methods for constructing programs, specific strategies for program segments etc. It hypothesises program design as a repeated process of problem understanding, problem decomposition and solution elaboration.

Problem understanding produces in the working memory an internal representation of pertinent relationships for the solution of the problem.

During problem decomposition two distinct approaches are hypothesised, namely "data-driven" and "process-driven". These respectively correspond to the situation when designers primary focus is either on the data specification or processing requirements. Two closely associated strategies of general applicability namely process next item and incremental design are also hypothesised. The first strategy corresponds to self-elaboration (i.e. how would I carry out the task to be performed?) and bears close resemblance to Hoc's (1977) notion of mental execution of the program, which his findings suggest influences program construction. The second strategy is one in which familiar and immediately achievable sub-goals are isolated, to formulate a partial solution and the remainder of the design subsequently made to conform. Its character can be summed up as "do what you can and fit the rest around it."

Solution elaboration is a process of composition that involves categorising components into appropriate clusters and allocating these clusters to the existing decomposition structure. During this process action clusters are retrieved from long term memory or generated in working memory and then allocated. In the former case in either their original or re-categorised form. In the latter case the generation process involves creating new clusters or combining existing clusters in a novel way. The two cases respectively corresponding to situations when the solution to the sub-problem is "known" or "not known". Siddiqi's notion of action clusters appears to bear a close similarity with Rist and Soloway's idea of programming plans.

The third model is based on Kant's (1985) empirical study which involved gathering protocols from under-graduate/graduate students attempting to design an algorithm for the convex hull problem. The theoretical basis for this model is Newell and Simon's (1972) theory of problem solving which views the problem solver as an information processing system and considers problem solving to consist of a state by state search of the problem space. Kant in common with previous models characterises programmer behaviour in terms of:

- problem understanding which involves listing the pertinent properties, and considering reformulations of the problem;
- solution planning which involves formulating a kernel structure through the selection of a set of general ideas and the strategies available to a designer; and refinement or elaboration of the kernel structure which involves determining whether the chosen strategy is feasible for the particular problem. If so, the kernel idea is refined and then elaborated, otherwise an alternative kernel idea may be considered.

The characteristics of behaviour that occurs during the formulation and refinement of the kernel structure are as follows:

- trial execution of the partially specified algorithm which is used to discover new opportunities or difficulties in the design process. Two types of trial execution are suggested: on concrete data, and on symbolic exemplar. The former is often employed for simpler parts of the algorithm and the latter for the complex parts;
- verification that the structure meets its specification which means initially attempting to solve some representative part of the problem and ignoring specific "details" such as base cases, initialisation, boundary conditions, degenerate inputs etc. and then subsequent "completion" of the algorithm being carried out by testing for all the cases they are aware of and incorporating those so called details in the structure of the solution;

- solution evaluation which is considered by experts as a criteria for choosing from alternative refinements. Possible criteria are: efficiency, ease of modification.

The role of trial execution as observed by Kant is also a characteristic noted by Hoc and Siddiqi. In relation to sequencing of decisions she in common with previous works notes that algorithm development does not always proceed in an orderly top-down fashion. There is considerable exploration to assist the feasibility of a particular strategy by either backing up to a higher or refining to a lower level.

In summary, it would appear that all three works postulate program designer behaviour as involving problem understanding, solution formulation and solution elaboration. The influencing factors in the latter two are the use of mental execution and retrieval and/or generation of plans. Furthermore, design is performed in a piece-meal manner of formulating an "island of certainty" around what is known and fitting the rest around it. This characterisation formed the basis upon which the observational study, reported in the next section, was carried out.

3. Observational Study

Aim

To investigate the presence of certain identified characteristics of behaviour of experienced programmers during software development.

Subjects

The five subjects were computer science lecturers, all with more
than eight years of experience in teaching programming.

Materials

The programming problem chosen was to print in a required format the representation of a bridge hand in suit and descending rank order and print the points value of the hand. The problem was chosen because it involved subjects in having to select both a suitable data and algorithm representation.

Method

Prior to each experiment each subject was given a set of oral instructions as explained in the necessary instructions for performing protocol gathering experiments. The usual precautions as outlined in Ericsson and Simon (1984) were followed for conducting the experiments. This meant adopting a balanced approach to intervention by not leading the subject or needlessly disturbing their reasoning processes yet at the same time through appropriate questioning eliciting the subject's 'cognitive processes'. At the start of each experiment (approximately one hour duration) the subject was supplied a written description of the problem and asked to design a program using pen and paper to document their development in a PASCAL like notation.

Analysis Of Data

Transcription of the video tape recordings together with the subject's notes provided the raw data on subject's behaviour. The division of spoken and written information in to protocols was made according to two rules. First, a break was made whenever the subject was judged to have paused in speaking or writing. Second if the speech or writing was relatively continuous, breaks were made between major clauses. The segmentation of lines into protocols were, thus, intended to give a rough indication of the conceptual units used by subjects. To distinguish verbal and written protocols in the transcriptions, the former were enclosed between quotation marks; and the latter were enclosed inside squares.

4. Discussion of Results

As mentioned above, the bridge hand problem was chosen because it involved subjects in having to select both a suitable data and algorithm representation. Different choices of data representation leads to need for different algorithm representation and this resulted in subject spending some time evaluating different possible representation. Basically the choice of representation for the hand is between either a two dimensional boolean array or the more obvious choice of either a one or two dimensional array of integers. The first representation, although less obvious as discovered in our experiment, leads to the need for a less demanding algorithm.

In the three models reviewed the issues concerning data representation is of secondary consideration to that of algorithm representation. Therefore, whilst it was possible to match segments of the protocol transcription to the processes suggested in the models, namely: problem understanding, solution formulating and solution elaboration, other issues not addressed by the models were also observed. The results are as follows:

4.1 Problem Understanding

In accordance with the three models, this behaviour manifested itself either as subjects writing statements that clarify the problem; questioning problem interpretation; referring to knowledge other than that about programming to determine limits or conditions of the problem. These processes seemed to aid the subjects to create a mental model of the problem under the investigation. The major part of problem understanding phase was considered to be completed when the subjects started the design phase, however, all subjects had to return to the problem specification, from time to time to obtain specific details of the problem. The example protocols are:

1) One subject questions the output format:

"How fuzzy are you when it actually says things like Clubs, Diamonds, Hearts, Spades in that order?"

2) Another subject asks:

"The representation can be anything. Can't it? Obviously we need to store suit and rank of each card."

3) A third subject's protocol for listing the important bits are:

"we have ..."

I-------------------------------------------I
!((Clubs), (Diamonds), (Hearts), (Spades))!
I-------------------------------------------I
"... and we have ..."

I-------------------------------------------I
!A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2!
I-------------------------------------------I
"... Alright that's input. "... let's see. Two separate things are required ..."

I-------------------------------------------I
!1) points card, !
!2) ordering for display. !
I-------------------------------------------I
"... Alright. Where should we go from here?"

One way of providing a possible detailed description of the underlying process in this behaviour is to use Hayes and Simon's theory of text understanding (1974) which suggest that this process involves alternately extracting information and building internal representations. However, this theory does not address the issues such as whether a graphical representation helps in problem understanding. Carroll et al. (1980) observed that people have more difficulty in understanding problems that do not lend themselves to graphical representations. Although such issues are a possible direction for further research, the nature of the task undertaken makes it difficult to focus on them or to obtain the type of evidence that would provide a richer theory.

4.2 Solution formulation and elaboration

In order to illustrate the trace of characteristics identified by
three models we present example protocols from a continuous segment of one of the subject's protocols. The subject has made the obvious choice of representing the hand as a thirteen element array of numbers and he is focusing on the central problem of designing a sort routine. This stage of program development begins by the subject writing:

```
!----------------------------------------------------------------------
!PROCEDURE SORT;
!{SORT HAND array into Descending order}
!----------------------------------------------------------------------
```

This corresponds to the subject specifying the routine to be designed. He then stated:

"... Well, I am going through every element up to 13, finding the smallest and pushing it into the bottom, so ...
"

This outlines subject's general strategy which in Kant's terms is the formulation of a kernel structure or in Siddiqi's terms as adopting a data-driven approach for a known sub-problem. He then wrote:

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N := 12</td>
</tr>
<tr>
<td>FOR I := 1 TO N</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
</tbody>
</table>
```

It is clear from the above written protocol that the subject has recognised, in terms of Rist's model, the need for a low level programming plan and attempts to retrieve it. Further evidence that he is attempting to remember or retrieve an existing plan is indicated by the following which was stated after a short pause:

"... I have done something like this before, so I am trying to remember how I did it in the past ...
"

The next stage of algorithm development begins with the subject saying:

"... Basically what we want is to sort the smallest element to the bottom, by comparing the ith element with i + 1th (element). ... once we got to the bottom, then we go back to the top again and then we sort the next smallest to the next to the bottom (position). ... In other words we decrease N by one and repeat our loop again".

This can be explained in Siddiqi's terms as problem decomposition based on mental execution of the task. He has established an explicit non-programming plan or kernel structure for the task. The subject then said:

"... Should I do the inside or outside (of the loop)"

He recognises the need for an inner and outer loop and then gives consideration to the generation and/or retrieval of a programming plan or action cluster.

After a short pause he wrote:

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IF HAND(I) &lt; HAND(I + 1)</td>
</tr>
<tr>
<td>THEN BEGIN</td>
</tr>
<tr>
<td>TEMP := HAND(I);</td>
</tr>
<tr>
<td>HAND(I) := HAND(I + 1)</td>
</tr>
<tr>
<td>END;</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
</tbody>
</table>
```

The written protocol shows that he has retrieved the "known" programming plan for swapping two elements. This tends to suggest the adoption of a "do what you can" strategy which forms the inner part of the loop. He then says and writes:

"... I've dealt with swap. That will swap one element...
... Let's put a FOR loop on that?

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR J := 1 TO N</td>
</tr>
<tr>
<td>DO</td>
</tr>
<tr>
<td>IF HAND(I) &lt; HAND(I + 1)</td>
</tr>
<tr>
<td>THEN BEGIN</td>
</tr>
<tr>
<td>TEMP := HAND(I);</td>
</tr>
<tr>
<td>HAND(I) := HAND(I + 1)</td>
</tr>
<tr>
<td>END;</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
</tbody>
</table>
```

This suggests that he is attempting to use a "fit the rest around it" strategy. He then said:

"... So far, I can swap one element to the bottom...
... Keep on doing this job until I run out of elements...
... I have to see that through .... Right, let's get another scrap paper...
"

This stage starts what Kant characterises as mental execution using concrete data (i.e., a random set of data). Briefly, this means that the subject then continued to perform trial execution by using specific example data sets until the algorithm was completed. This process results in the subject discovering the need to decrement N, change the array index from "I" to "J" in his inner programming swap plan, and also that the upper limit of "J" should be "12" not "N".

Having interpreted a major segment of one subject's protocol in terms of the three models discussed in section 2, we can see their similarities. The following section briefly discusses the limitation of the models by pointing out some of the characteristics observed in our study which are not adequately addressed by them.

5. Comparative Issues

The individual differences in approaches taken to the problem were substantial in that from the five experimental subjects four different approaches were observed. Two subjects chose to represent the pack of cards as a two dimensional boolean array of 4 by 13 elements, whilst one subject chose a one dimensional integer array between 1 and 52, one chose four one dimensional character arrays and the last subject chose a two dimensional integer array.

Given that the problem is neither complex nor large it is significant that there was such diversity in approaches. The
works presented above do not report such diversity although in Siddiqi's work two distinct approaches are hypothesised. However, other studies confirm our observation. For example Curtis (1986) reports that, even with professional programmers where one might expect a significant degree of homogeneity, they exhibit a variety of approaches on designing a relatively simple program.

In our protocols searching for a method (i.e. algorithm design) and searching for a suitable data representation were carried out in parallel. There was a constant shift in the focus of attention from data representation to algorithm design and vice-versa. For example one subject considered two different data structure for the input and assessed the difficulties of finding suitable algorithm to implement each. This evaluation was the basis for subsequent decisions in design:

"... The input is 13 cards. Now they can either be passed in (to the program) as for example an ordered pair of suit name and value or it could just be that each card comes in as a number between 1 and 52..."

... The first one is easy to handle. The second one is more realistic if you use something like FORTRAN."

The first part focuses on enumerating possible choices for data representations and the second part focuses on considerations for algorithm design and evaluation of the two possible data representations.

The three models presented do not address themselves to this consideration although Kant does suggest that consideration of appropriate data structure can be relegated to the program synthesis stage. In the bridge hand problem and for many other problems, the choice of an appropriate data structure at an early stage of design is, in our opinion, an essential component.

It would appear that there is a significant similarity in the explanation offered in the three models. In the most general terms they view the programming process as consisting of understanding the problem and the formulation and elaboration of the solution. They also advance the view that programming is a process involving refinement and synthesis. Moreover, whereas they acknowledge that the flow of decision making appears top-down (particularly when the sub-problem is "known" or when there is a retrievable non-programming plan) there is considerable hierarchical planning. In particular, Rist's notion of selecting a focal idea and building a plan around it closely resembles Siddiqi's idea of incremental design strategy, and Kant concept of solution planning around a kernel idea. Similarly, the importance of what Hoc terms as mental execution appears as self-élaboration or process next item in Siddiqi's work and as trial execution in Kant's work. Rist's notion of a plan and Siddiqi's idea of an action cluster form complementary concepts, particularly the respective correspondence between known and unknown clusters, and existing and novel plans.

The three models however focus on different aspects of program design. Rist's primarily describes the program design process in terms of the types of knowledge available to program designer. Problem decomposition in this model yields non-programming plans and solution composition retrieves programming knowledge in form of plan, however, insufficient explanation is provided on how the decomposition takes place. Whilst Siddiqi's model provides an explanation of the design process by focusing on the factors governing the choice of decomposition strategy, however, it does not provide an adequate explanation of programming knowledge. Kant's model provides an explanation for designing a complex algorithm and therefore identifies certain behavioural characteristics not considered by Rist and Siddiqi. Our study provides numerous examples of protocols that concur with behavioural characteristics advanced by these models and also reveals characteristics not reported by them. In particular the issue concerning the choice of an appropriate data structure at the early stage of design and the consequences of that choice on subsequent decision making.

6. Conclusion

From our perspective because of the similarity in the explanations offered in many of the empirical studies of programmers, there is a need to consolidate the results of these works into a unified model. Indeed, the present work which advances a relatively simple characterisation, presented in the discussion, and which forms the basis for protocol analysis is the first step in that direction. In particular, a further step to express models in operational rather than descriptive terms, thereby improving their potential for prediction, has been the characterisation of our earlier results in terms of a blackboard architecture (Siddiqi, Sumiga, 1986).

Finally as Curtis (1986) points out there is an urgent need to "contribute timely knowledge about the crucial issues in software development" so that our "findings can be used to guide new programming technology". For example an obvious guideline of this work in relation to knowledge-based support tools for software development is that they should not enforce a top-down regime but should facilitate incremental design and provide tools that assist the designer's natural tendency to use mental execution.

In our opinion, if more guidelines are to be identified it is essential that empirical research places more emphasis on providing evidence that supports, or refutes much of what has already been discovered, rather than attempting to offer what appears to be new explanations. The danger of not doing this is as Curtis strongly argues that one pays the penalty of discovering that:

"Many of the results obtained are not surprising, since they represent the demonstration in programming of cognitive phenomena that had already been demonstrated in other areas."
References


[2] Curtis, B. (1986), "By the way, Did anyone study any
real programmer?", In proceedings of the conference on
empirical studies of programmers, Edited by Soloway E.
and Iyengar S. Ablex publication.

Verbal Reports As Data., MIT press, Cambridge, MA.

ten Problem Instructions.", In Gregg, L.W., Ed., Know1

A General Framework For Problem Solving?", Technical
Report, Huristic Programming Project, Computer
Science Department, Stanford University.

Model Of Planning', Cognitive Science, No. 3.

Learning a Programming Language.', International
Journal of Man Machine Studies, 9, 87-105.

Knowledge Based Program Understanding", IEEE
Trans. On SE. Vol SE-11, No. 3.


model', (Draft Copy).

[12] Rist, R.S., (1986), 'Plan in Programming: Definition,
Demonstration and Development', In proceedings of the
conference on empirical studies of programmers, Edited
by Soloway E. and Iyengar S. Ablex publication.

Design.', In proceeding of conference on empirical
studies of programmers, Edited by Soloway E. and
Iyengar S. Ablex publication.

MA: Cambridge University Press.

Problem Decomposition Strategies Used In Program
Design.' Ph.D. Thesis, University Of Aston In Birming
ham.

Behaviour', Proceeding of Conf. of BCS, HCI Specialist
Group, edited by F. Johnson and S. Cook.

of a Proposed Model of The Program Design Process',
The Fourth Symposium on Empirical Foundations of
Information and Software Sciences, Atlanta, Georgia,
USA.

Programming Knowledge.', IEEE Trans. SE, 10(5),595-
609.