ment," USC/Inform. Sci. Inst., CONSUL Project Internal Design Note.

Robert Neches received the Ph.D. degree in psychology from Carnegie-Mellon University, Pittsburgh, PA, in 1981.

He then spent a year as a postdoctoral fellow at the Learning Research and Development Center of the University of Pittsburgh. His research in machine learning produced a system called HPM that emulated developments in children's mathematical skills, and (with P. Langley) a language called PRISM which has been used in a number of machine learning projects. Since joining Information Sciences Institute, Marina del Rey, CA, in 1982, his research and publications have concerned both expert systems and artificial intelligence applications to friendly computing environments.

William R. Swartout received the B.S. degree in mathematical sciences from Stanford University, Stanford, CA, and the M.S. and Ph.D. degrees in computer science from the Massachusetts Institute of Technology, Cambridge.

He is a Research Scientist at the University of Southern California Information Sciences Institute, Marina del Rey, CA, where he was the principal designer of two systems for making program specifications more understandable, the Gist Paraphraser and the Behavior Explainer. He is currently Project Leader of the Explainable Expert Systems project at ISI, developing a framework for creating expert systems that makes them more understandable and maintainable. His research interests include expert systems, natural language generation, and program synthesis.

Johanna D. Moore received the B.S. degree in mathematics/computer science and the M.S. degree from the University of California, Los Angeles, in 1980 and 1982, respectively. Her Master's thesis dealt with the issue of transaction management in a distributed operating system.

She is currently working toward the Ph.D. degree in artificial intelligence at UCLA and the University of Southern California Information Sciences Institute. Her research deals with planning explanations, natural language generation, and user modeling.

The Role of Domain Experience in Software Design

BETH ADELSON AND ELLIOT SOLOWAY

Abstract—**A designer's expertise rests on the knowledge and skills which develop with experience in a domain. As a result, when a designer is designing an object in an unfamiliar domain he will not have the same knowledge and skills available to him as when he is designing an object in a familiar domain. In this paper we look at the software designer's underlying constellation of knowledge and skills, and at the way in which this constellation is dependent upon experience in a domain. What skills drop out, what skills, or interactions of skills come forward as experience with the domain changes? To answer the above question, we studied expert designers in experimentally created design contexts with which they were differentially familiar. In this paper we describe the knowledge and skills we found were central to each of the above contexts and discuss the functional utility of each. In addition to discussing the knowledge and skills we observed in expert designers, we will also compare novice and expert behavior.**

Index Terms—**Artificial intelligence, cognitive models, cognitive science, software design.**

I. Introduction: Motivation and Goals

A DESIGNER'S expertise rests on the knowledge and skills which develop with experience in a domain. As a result, when a designer is designing an object in an unfamiliar domain he will not have the same knowledge and skills available to him as when he is designing an object in a familiar domain. In this paper we look at the software designer's underlying constellation of knowledge and skills, and at the way in which this constellation is dependent upon experience in a domain. What skills drop out, what skills, or interactions of skills come forward as experience with the domain changes? Specifically we ask the following.

- *What happens when well-known information must be used in novel ways?*
- *What happens when domain experience cannot be relied upon? What are the general knowledge and skills which remain?*
- *What happens when the object has been designed before?*

To answer the above questions, we studied expert designers in experimentally created design contexts with which they were differentially familiar. In this paper we describe the knowledge and skills we found were central to each of the above contexts and discuss the *functional utility* of each; what role it played, how it contributed to getting the job done. In addition to discussing the knowledge and skills we observed in expert designers, we will also compare novice and expert behavior. The comparison points out the utility of the knowledge and skills of the expert.

Manuscript recieved June 3, 1985; revised July 8, 1985. This work was supported by the National Science Foundation under Grants IST-8505019 and IST-8519255.

B. Adelson is with the Department of Computer Science, Yale University, New Haven, CT 06520, and the Division of Information Science and Technology, National Science Foundation, Washington, DC 20050.

E. Soloway is with the Department of Computer Science, Yale University, New Haven, CT 06520, and Compu-Teach, Inc., New Haven, CT 06520.

0098/5589/85/1100-1351$01.00 © 1985 IEEE

Organization

In the next section we discuss our methodology and the nature of the data it produced. Following that, in Section III, we discuss the behaviors found in each of the contexts in which we observed our *expert* designers. We will also discuss how the behaviors changed with context. In Section IV we compare expert behavior to novice behavior. We close this paper with a discussion of the implications of our findings.

II. Methodology and Resulting Data

Here we describe our methodology, protocol analysis, and a description of the type of data it yields.

A. Protocol Analysis

For our protocol analyses individual designers were given a problem of theoretical interest to solve. A protocol was taken of the solution process, forming a video taped record of the designer's words, actions, gestures, and facial expressions. This methodology is based upon the techniques developed by Newell and Simon and their colleagues [7], [11]. During the problem solving session the experimenter remained with the designer, encouraging "thinking aloud," but taking care not to influence the course of the design; the designers were asked "What are you thinking now?" *not* "Are you thinking about x?"

After the protocol was obtained, it was analyzed by the experimenters. A careful distinction was made at this point. Of interest was what the designers *did*, rather what they *said* they did. In accord with this caveat, the tapes were analyzed for regularities in the designers' behavior. We were looking for behaviors done by all of the designers or for behaviors done repeatedly during a single design. Identifying these regularities in the design process allows us to build a picture of what is important in design.

B. The Designers

Experts: Three expert designers were studied, E1, E2, and E3. All had at least eight years design experience, were employees of ITT and were expert in the design of communications systems. Table I shows the tasks done by each expert designer.[1]

Novices: Two novice designers were studied. Both had worked for several years as programmers and for less than two years as designers. Both were employees of ITT, working on the design of communications systems. Table I shows the tasks done by each novice designer.

C. Procedure

On each design task, one of three design specifications was given to the designer. The three design specifications are shown in Figs. 1, 2, and 3. The designer was video taped while working out a design that could subsequently be given to a programmer. The environment in which the protocols were collected was the regular work environment of the designers.

[1]Due to the lack of availability of professional designers as participants in experimental studies, not all our tasks were done by all our designers.

TABLE I
Task Done by Each Designer

Designer:	Task: EMS	LS	IH
Expert 1	x	x	x
Expert 2	x		
Expert 3	x	x	
Novice 1	x		
Novice 2	x		

```
Design an electronic mail system (EMS) around the following
commands:

READ, REPLY, SEND, DELETE, SAVE, EDIT, LIST-HEADERS.

The mail system will run on a very large, fast machine so
hardware considerations are not an issue.
```

Fig. 1. Domain = Familiar × Object = Unfamiliar.

```
Design a Library Record-keeping System (LS)
around the following:

COMMANDS:
LOAN, RENEW, RETURN, HOLD, RECALL,
NOTIFY PATRON (of overdue book, etc.), CALCULATE FINES.

PATRON TYPES:
General, Special patron, Cooperating library.

BOOK TYPES:
Best sellers, General listings, Rare books.

The LS will run on a multi-user mainframe. It is very large and fast, so
computational limitations are not an issue.
```

Fig. 2. Domain = Unfamiliar × Object = Unfamiliar.

```
Design an interrupt handler.

  • The processor will be an 8086.

  • The only device will be an INTEL 82586. (Documentation was provided)

  • The interrupts will result from the chip's communication with ETHERNET.
    (Documentation was provided)

  • The operating system is UNIX.
```

Fig. 3. Domain = Familiar × Object = Familiar.

D. Tasks as Representative Design Contexts

Each of the following research contexts was designed to be representative of one of the major types of contexts in which designers work. We investigated this representative set of contexts in an attempt to observe how the skills and knowledge necessary to design would vary in each one. In order to create a full set of design contexts we systematically varied the familiarity of the following.

1) The domain of the object to be designed. The domain of the object corresponds to the general classification of the system according to its use. For example, we consider the electronic mail system described below to be in the domain of communications systems. Familiarity was varied by picking two clearly separate points along a familiarity continuum for *our* designers. We defined an *unfamiliar* domain as one in which the designers had never

designed before. It was not so unfamiliar as to prevent them from understanding the task. For example, the designers knew how the library system described below would work. They had not, however, designed or used them and had not worked as librarians. We defined a *familiar* domain as one in which the designers had designed before, even if they had not designed the *particular* object described in the specification.

2) The object to be designed. Familiarity was varied here by picking an object that either had or had not been designed before.

As we see in Table II, varying the familiarity of the object and the domain gives four potential design *contexts*.

1) Domain=Familiar × *Object=Unfamiliar.* This is probably the context in which designers most frequently find themselves. In this context the designers are working in their domain of expertise, in order to create something new. Because the domain is familiar and the object is unfamiliar this context allows us to ask:

- *What happens when well-known information must be used in novel ways?* For our designers the task representative of this context is shown in Fig. 1.

2) Domain=Unfamiliar × *Object=Unfamiliar.* Because both the domain and the object are unfamiliar this context allows us to ask the following.

- *What happens when domain experience cannot be relied upon?*
- *What are the general skills/knowledge which remain?*

For our designers the task representative of this context is shown in Fig. 2.

3) Domain=Familiar × *Object=Familiar.* This context allows us to look at a major component of expertise, dealing with already known aspects of a problem. This context allows us to ask:

- *What happens when the object has been designed before?*

For our designers the task representative of this context is shown in Fig. 3.

4) Domain=Unfamiliar × *Object=Familiar.* No task exists which is representative of this context. Therefore, data on this context were not collected.

The design tasks were chosen to be ones which would actually engage the designers, in order to bring out realistic design behavior. Additionally, the level of detail of the problem specifications were equivalent and the tasks were designed to take approximately equal amounts of time to complete.

III. Expert Designers in Each Design Context

Nature of the Results

For each of the tasks it took approximately two hours to work out a high-level design. The designs obtained from the experts were clearly judged to be good and bug-free except, not surprisingly, in the case of the Domain=Unfamiliar × Object=Unfamiliar context. Here one expert had two oversights which were not detected because he could not use the domain specific skills which had prevented this kind of oversight when he was designing in his domain of expertise. This will be discussed in Section III-B. The designs obtained from the novices were clearly serious but not bug-free attempts.[2] The behavior of the novices will be discussed in Section IV.

TABLE II
Tasks Representative of Each Design Context

		Object: Familiar	Object: Unfamiliar
Domain	Familiar	Interrupt Handler	Electronic Mail System
	Unfamiliar	—	Library System

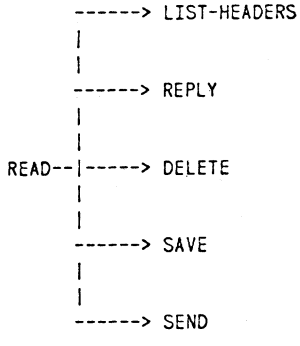

Fig. 4. E2's diagram of the EMS.

In what follows we present our observations on how domain experience influences what knowledge and skills can be used in each of the design contexts we have studied.

A. Designing an Unfamiliar Object in a Familiar Domain: The Electronic Mail System

Here the designer is working within his domain of expertise on a never-before-designed system. This context will allow us to ask:

- *What happens when well-known information must be used in novel ways?* We present this context first because it is probably the most frequently occurring context. It can therefore, act as a sort of base case and form a framework for the reader. We will present the observed behaviors first and follow with discussions of the role of experience, each behavior's utility, and the interactions among the behaviors.

1) Behavior—Formation of Mental Models: All of our expert designers seemed to form an internal working model of the design-in-progress. We are defining a model as a representation which is capable of supporting *mental simulations* of the design in progress. It is a representation in which the commands of the system being designed can be viewed as actions. Further evidence of the existence of these internal models is that frequently they served to generate external models in the form of sketches of a design-in-progress. Typically these sketches consisted of some initial node with arcs from it to other nodes in the sketch (Fig. 4). The initial node depicted an action or command

[2]Note that in a research context we are interested not so much in bug-free performance as in serious attempts. In studying the design process both the errors and the successes are of interest.

of the system and the arcs specified the actions that could follow as a result of the initial action.

In the following quote from E3 we see an example of a mental model and how it allows the designer to simulate the actions of the system. Here E3 is inspecting his most recent sketch of his design-in-progress by simulating an aspect of the design's functioning. He considers the recognition of input to the system.

> "There's some stream of input come in. I've found the key word 'mail' embedded in there ...
> So, having done that, presumably in going from one end of the string ('mail') to the other, I've also found other strings of characters which I haven't analyzed them for what this means yet, but I've got them separated out ..."

In the above quote we see that the designer has a mental model of the mail system that allows him to simulate an aspect of the design-in-progress.

2) Behavior—Systematic Expansion of Mental Models: We observed that the nature of each designer's mental model changed as his design progressed. Each designer's model started out at an abstract level of representation and progressed to a concrete one. This is illustrated by the following two quotes from E3. Early in the task E3 stated:

> "What we have here is information flowing through a system."

At 20 minutes into the task E3 made the statement quoted earlier in which he considered how input to the system would be parsed into tokens. Contrasting the two quotes allows us to see the progression from abstract to concrete. Both are accurate representations of a mail system; however, the quote about token recognition reflects a much more concrete model than does the quote about information flow.

Additionally, only one level of representation is focused on at a time and each level of representation is only a bit more detailed than the next. These two aspects of systematic expansion are illustrated by the following quote from E2. Here E2 had just drawn a hierarchical diagram of the commands of the EMS and then, pointing to the top level of the diagram, he said:

> "This (drawing) will help us structure our solution to the problem at a higher level ...
> Then we will go into each one of the building blocks that help us write the processing step at each step in the state diagram."

As we can see here, E2 is picking one level of abstraction to focus on at any one time.

3) Behavior—Simulation of Mental Models: We observed all of our subjects repeatedly conducting simulation runs of their mental models [8], [9]. The following quote provides us with an example of an expert designer running a simulation. E2 has just drawn a sketch in order to represent to himself *all* the actions that the EMS can take from the current state. Here he traces through the arcs of the drawing (Fig. 4) as he considers the functioning of his EMS-in-progress.

> "What we can see here is one state (accessing) several other states and after the operation is completed, control of the state transition going back to the initial state."

4) Behavior—Representing Constraints: Above we described the mental model, the representation which supports simulation. It seems there is a second kind of representation in which the commands of the system are not represented as actions which support simulations [10]. Rather they are represented in a way that supports operations such as the assertion of properties, the deduction of implications, etc. This sort of representation does not directly contain the specific causal relationships which are necessary for a simulation, we therefore find it useful to think of it as a separate representation.[3] The following quotes from E1 illustrate the operations which this representation supports:

> *Assertions:* We must be able to prepare, send, receive. In receiving, a number of choices and decisions have to be made as to the disposition of it.
> *Implication:* The system must be able to store them (mail messages). The system must be able to handle abnormalities throughout it.

In the above two quotes we see E1 attribute several properties to the EMS. This is done at the beginning of the design session. As discussed below, the designer makes these assertions early on in order to constrain the initial problem statement. This is done so that the designer has a sufficiently specified conceptualization of the problem in order to form a mental model.

5) Behavior—Retrieving Labels for Plans: In the following quote we see E1 state that objects such as a message store will exist in the system. He does not, however, concern himself either with working out or recalling the implementation details at this point. We see this kind of statement in situations in which the designer has previously worked out a solution to an aspect of the problem. In these cases he knows that he already has an answer to this subproblem; it exists as a plan in memory to be retrieved when systematic expansion allows.

> "For data there must be some data of the users: the destinations, the addresses, data of update formats.
> ..There must be the message store itself.
> The message store would come in several forms:
> The store for messages sent but not received.
> The store of messages received by some individual.
> I think of UNIX, the dead letter concept ... the store of messages that we can't do anything with."

In the above quote E1 is enumerating the *labels* for elements of the problem which have a solution stored as a plan in memory. The plans themselves will be retrieved later when the level of detail of the design matches the level of detail of the plan. Before that time the label is retrieved as a place holder for the plan. Here we have a second representation with a role other than simulation.

6) Behavior—Note Making: Repeatedly we saw the ex-

[3]The representation may allow the causal relations to be inferred, but they are not present before these inferences are made.

pert designers stop themselves just as they were about to develop a concern that caught their eye. These concerns had to do with constraints, partial solutions, or potential inconsistencies which were not at a level of detail appropriate to the current mental model. When the designers would stop themselves they would then make a mental note (or an actual, written one) as a reminder. We also observed that our experts would return to these notes at an appropriate time.

The following quote from E1 illustrates the Note Making process.

> "At this stage, I need the functional specifications ...
> *I'll come back and do that in a little while, time permitting, ...*
> and carry on making some other assumptions here.
> Store ...Read ...OK. That's good enough"

In the above quote E1 stops in midsimulation and starts to consider the functional specifications underlying his current model. He then stops himself and makes a note and in the last sentence returns to his simulation of STORE and READ.

Discussion—Functional Utility of the Behaviors and their Interactions: Here we discuss why each behavior is useful in this particular context; how it helps in the design of the object, and how it helps in relation to the other observed behaviors.

- The *mental model* can be thought of as the designer's internal design-in-progress. Its function is to allow the designer to choose a particular way of thinking about the problem and out of that way of thinking to generate a working model to elaborate upon. We see evidence of this in the data presented above for the formation and simulation of the model.
- There are several things to note with respect to the function of *simulation*.

1) When designing an object the goal state is specified as a behavior; the designer knows what a functioning EMS should do. The current state, however, is *not* specified as a behavior, it is specified as a set of partially designed mechanisms which will eventually exhibit the behavior. The designer knows, for example, that he needs a mechanism called READ but he is not sure that his READ-in-progress will actually do what it is supposed to, in a bug-free way. The simulation of the mental model allows the current state, to be specified as a behavior. This in turn allows the current state to be compared to the goal state which is already specified as a behavior. This enables the designer to solve the problem by assessing the nature of the differences between the current and the goal states and then choosing a means to reduce that difference. This is referred to as a solution through means-ends analysis [11].

2) Recall that the mental model goes through a progression from an abstract to a concrete state. Recall also that our designers were used to building communication systems, but that they had never built an EMS before. That means that in building an EMS the designers are combining familiar processing components into a novel assembly. As a result the designer has to check for unforeseen interactions; he must check for the presence and consistency of all of the possible inputs to each command in the EMS-in-progress. Simulation allows this check to take place by providing the designer with all of the inputs in question. It seems that this is one way in which familiarity with a domain contributes to design expertise. Here, where the designer has familiarity with the domain but not the object, he is able to use his domain knowledge to build a respresentation and simulate it. That this is a result of domain knowledge becomes clear when we contrast this design context with the next context in which domain knowledge is absent.

3) Once simulation is needed *systematic expansion* is needed since the I/O of all elements in a simulation must be at the same level of description. The following example illustrates this point. If element A's output is not as detailed as element B's input, element B will not be able to use the output from element A or will have to do additional work first. This is also true if element A's output is more detailed than element B's. Therefore, it is systematic expansion that allows simulation to function smoothly.

- Once systematic expansion is needed *note making* is needed as well; concerns arise during simulation that are at the wrong level of detail. Dealing with them immediately would lead to commands being defined at different levels of detail and this would violate the requirements of later simulations. The concerns which arise during a simulation need to be post-poned until the appropriate time; note making allows concerns to be postponed and then returned to.
- The assertions and implications presented above serve to *constrain* the designer's mental representation of the problem. They help him decide what properties to add to his representation when he feels that it is insufficiently specified. This is useful because there is evidence that a *runnable* mental model cannot be constructed until the design has reached a sufficient level of specificity [6], [12]. The making of these assertions is prominent in the early parts of the EMS task, as the designers attempt to give their first, most abstract, mental models sufficient specificity.

In the next section we present the behaviors we observed in a context in which an expert designer designed a never-before-designed system outside of his domain of expertise.

B. Designing an Unfamiliar Object in an Unfamiliar Domain: The Library System

The present context allows us to address the issue of how the design task changes when both the domain being designed in and the object being designed are unfamiliar. Because both the domain and the object are unfamiliar this context allows us to ask:

- *What happens when domain experience cannot be relied upon?*
- *What are the general knowledge and skills which remain?*

First we present the behaviors observed, then we discuss and compare these behaviors to those found when the de-

signer was working on a novel object in his domain of expertise.

1) Behavior—Formation of Mental Models: As in the EMS, the designers again formed a mental model of the design. However, their model of the LS never became as detailed as their final model of the EMS. Additionally, the model of the LS did not contain some sorts of information that the model of the EMS did. In the LS the designers represented information about how the system would appear to a user. However, they did not address questions about how the system would be implemented. For example, in the LS the designers did not address the issue of how the system would interact with elements of a computing environment such as the operating system. In the EMS they did. In neither task was the operating system specified as something to be concerned with, but in the EMS the designers chose to consider this kind of detail, in the LS they did not.

2) Behavior—Systematic Expansion: Once again the expert designers chose one level of abstraction at a time and then worked at the level until they were satisfied. For example, at E3's first level of detail he simply specified that there would be a patron database, a book database, commands operating on the patron database, and commands operating on the book database. Next he developed the LS by defining both the set of commands in the problem statement and the book database. Notice that this is an error in systematic expansion; the patron database was never expanded. Below we discuss why we may expect errors in systematic expansion in an unfamiliar domain.

3) Behavior—Simulation: In the following quote, the representation of the commands as actions again suggests a simulation process. Here, when considering RECALL, E3 says

> "We need to pull out the record . . .
> See who else wants it . . .
> Send out the letter saying "Please bring it back."

Not only is E3 regarding the commands as actions to be performed, additionally, he is stating the pre- and post-condition for the action; two hallmarks of simulation. However, in this task, unlike in the EMS, simulations of commands were performed in isolation, not in the context of the effects of all the other commands. We see that only *local* simulation was done in the following quote where E3 discusses the input to RECALL. Below E3 decides that RECALL needs a subroutine or "background program" to pass it input information:

> " . . .and probably you want some sort of background program that runs on a daily basis that does some sort of audit of your records so that it can run out a daily log of over-due books, books that have been requested and haven't come back in, or, in cases of oversight, books that have come back in and no one has noticed it.
>
> Maybe someone just logged it back in and not looked at the record.
>
> You need to track things like that, so you need to do that every morning. What's the current state of affairs, what do you need to do today."

Had E3 been doing a global simulation he would have noticed that "logging it back in and *not* looking at the record" *and* having RECALL find that the book is marked as HELD could not occur since the RETURN command does both. E3 knew that RETURN yields both results because in the simulation of the RETURN command, which he had done three commands prior to this quote E3 noted that RETURN yielded both results. The subject's oversight suggests that he is doing accurate simulations of the individual commands, but that he is not doing a *global* simulation of the system as a whole.

The *order* in which the commands were designed also suggests a global model is not driving the simulations as was the case in the EMS task. In the LS both E1 and E3 designed the commands in the order in which they occurred in the problem statement shown in Fig. 2. This ordering is distinct from the ordering which would have occurred if the commands were being designed in an order reflecting a simulation run of the LS. In comparison, the order of occurrence in a running system *was* the order used in designing the more globally driven EMS task. Illustrative evidence that the order used in designing the commands of the LS does *not* reflect a run of the LS comes from E3. HOLD was designed right after RENEW, and RECALL was designed right after HOLD. In both these cases the first command is *not* a precondition for the second. In fact, in the first case the first command (RENEW) precludes the possibility of the second (marking the book HOLD). In the second case the first command (marking the book as HOLD) could not be a precondition for the second command (RECALL) since they actually need to happen in the opposite order (RECALL a book and then HOLD it).

4) Behavior—Representing Constraints: It was our sense that fewer constraints were added here than in the LS. Additionally, the constraints that were represented were different from those mentioned in the EMS. The constraints mentioned here never dealt with *how* the functions of the LS would be implemented, in the EMS they often did. For example consider E3's quote here:

> "The database has multiply-keyed records so you could have book title, book author, synopsis."

This quote is less concerned with implementation than, for example, E1's above quoted concern that "a number of choices" have to be made about the different ways in which a message can be disposed of after it is read.

5) Behavior—Retrieving Labels for Plans: It seemed that fewer labels for plans were retrieved here than in the EMS context. The few that were retrieved again seemed different from the ones retrieved for the EMS. Here the plans had to do with what properties the system would have. No statement was made as to how those properties would be implemented.

> "The first thing required will be a Dewey decimal system database."
> Having described the database all you've got is a need for, as far as I can see is two systems."

The above quotes from E3 illustrate that the designers'

plans did not have to do with how the LS system would be implemented.

6) Behavior—Note Making: Interestingly, note making was absent, although its presence might have saved E3 from forgetting to develop the patron database. Why note making is absent in this context is discussed below.

Discussion—Differences between Designing an Unfamiliar Object in a Familiar Domain and an Unfamiliar Object in an Unfamiliar Domain: Here we present comparisons between the behaviors we saw when the domain of the object was familiar or not familiar. We also draw inferences about how designs are created when domain specific skill and knowledge is present or absent.

- With respect to *mental models:*

1) The lack of detail found in the model of the LS as compared to the EMS can be attributed to lack of domain experience. In both tasks the specific object being designed was novel and the operating system was not specified as something to be concerned with; therefore, the only difference in the two tasks was familiarity with the domain.

2) Additionally, in this context, where the domain is unfamiliar, a global model of the design was not formed. This suggests that the ability to form a global model is tied in with familiarity with the domain of the object being designed. How domain knowledge contributes to the formation of a global model is not yet clear.

- With respect to *simulation:*

1) In this design context the designers did not have experience designing in the domain. Because simulation occurred, this suggests that the skill of simulation is general rather than one which stems from familiarity with a specific domain. This may be the case because in order to conduct a simulation the designers need only to be able to represent the commands as actions which have effects on other commands; this can be done with only general knowledge about the result of each action.

2) In this context we found that the ability to form a global model can drop out while leaving the ability to simulate. This suggests that the two abilities are independent.

3) A global simulation is one which simulates all of the inputs to a command. However, it cannot be done without a single global model because one unified representation is needed to keep track of the many details attended to in a global simulation [2]. These global models are the result of domain experience [1], [6]. If, through experience, the designer has a single representation for keeping track he will use it, allowing a successful global simulation to be done. This was the case in the EMS. If he does not have sufficient experience in the domain, he will not have a single representation of the whole system. As a result, he will not be able to keep track of all of the details of a global simulation. This was the case in the LS.

- This context allows us to make two interesting observations about the nature of *systematic expansion:*

1) It appears that working at one level of abstraction at a given point in time is a skill which is used automatically in the sense that the designer attempts to do it even here where the global simulation that it usually serves is missing.

2) We also claim that systematic expansion is aided by global simulation; the global model used for the simulation includes all of the elements of the design; this sets the designer up to have and be aware of all of the elements that are then to be expanded. Our claim that global simulation aids systematic expansion is supported here. Global simulation was absent and an error occurred in systematic expansion; the database of library patrons was not expanded by E3.

- It is not surprising that *note making* was absent here. The designer does not have experience in this domain and therefore cannot recognize when he is in a spot in which the underlying details could potentially lead to trouble if they are not integrated with care at the appropriate time. Because of his lack of domain experience here, he cannot make a note to himself to watch out for potential trouble spots.

Behaviors which are found even when the domain is unfamiliar can be considered behaviors which are general to design. We observed several such behaviors. In the LS where the designers lacked familiarity with the particular domain they built a mental model and attempted to simulate it and expand it systematically.

We also see that there is some domain specific knowledge and skill which is necessary to getting the job done. In the LS, when domain specific knowledge and skill is absent a global model could not be built. This precluded the ability to do global simulation and led to an error in systematic expansion. Additionally, there was a decreased use of constraint making and retrieval of plans.

In the next section we present the behaviors we observed in a context in which an expert designer designed a familiar system.

C. Designing a Familiar Object in a Familiar Domain: The Interrupt Handler

The present context allows us to address the issue of design when both the domain being designed in and the object being designed are familiar. Because both the domain and the object are familiar this context allows us to ask:

- *What happens when the object has been designed before?*

First we present the behaviors, then we compare them to those found when the domain is familiar but the object being designed is not.

1) Behavior—Formation of Mental Models. The designer formed a mental model early on. Almost as soon as he began the design he drew a picture of the interrupt handler and its relation to the interrupting device (the Intel chip) and the operating system. However, the drawing was striking in that although it was very abstract it was, nevertheless, very well-formed. It was abstract and yet well-formed in that the designer was able to clearly represent the fact that the operating system and the interrupt handler would communicate without including any details about how the communication would be implemented.

2) Behavior—Systematic Expansion of Mental Models: Once the designer had drawn himself a picture of the interrupt handler in its abstract form, he turned his atten-

tion to the functionality of the chip. Although he had designed interrupt handlers in the past, this *particular* chip was new to him and so it seemed to catch his interest.[4] He considered the properties of the chip in some detail, although he did eventually stop himself and return to the point in the abstract model where he had been. This exploration of detail does differ from what we saw in the EMS where exploration was cut off sooner and postponed via the making of notes.

3) Behavior—Simulation of Mental Models: We found no simulation on the familiar parts of the task. This finding is discussed below. As in the EMS, simulation was done when elements had to be combined in a novel assembly. When the designer was specifying the features of the interface between the new chip and the interrupt handler we did see simulation.

4) Behavior—Representing Constraints: The development of constraints occurred late in the task, as the designer began to consider the functional properties of the chip, the one aspect of the task which was least familiar.

5) Behavior—Retrieving Labels for Plans: This behavior was strikingly frequent here. For this familiar task the designer had many plans in memory which would be retrieved at the appropriate time in order to create this design.

6) Behavior—Note Making: As discussed below, note making was absent here.

Discussion: Differences between Designing an Unfamiliar Object in a Familiar Domain and a Familiar Object in a Familiar Domain: Here we present comparison between the behaviors we saw in the IH and the EMS context. We also draw inferences about how designs are created when experience with the particular object being designed is present or absent.

- Early in the task the designer's *mental model* of the problem was clear enough to allow him to draw a sketch of the system without hesitation. This is not surprising in that the designer had designed his type of object before and, as a result, had probably drawn this drawing before as well.
- We can compare *systematic expansion* here and in the EMS. Exploration of ill-understood details were pursued here but not in the EMS. Perhaps the designer only allows himself these explorations when he has designed the object previously. This is the one time when exploration will least threaten systematic expansion. This is the case because if the mental model is lost from working memory it can be easily reconstructed if it has been constructed frequently in the past. As a result, details can be explored here and then the mental model can be reconstructed when the designer is ready to continue systematic expansion.
- We found a lack of *simulation* on the familiar parts of the task. This is not surprising since the designer already had plans for successful solutions to the well-understood parts of the design. These plans had been found correct by the designer on previous occasions. Therefore it was not necessary to simulate them in order to evaluate their correctness. For the parts of the design which were unfamiliar, no plans existed and simulation was done. In sum, we can make the following statements about simulation:
- Global simulation is an effective tool, as we saw in the EMS.
- It cannot be used in the absence of domain knowledge, as we saw in the LS.
- It will not be used when a previously developed plan exists, as we saw in the IH.
- The lack of *note making* in this task is not surprising in that the details of the design had already been worked out. The details did not need to be noted since the designer already had a successful plan for dealing with them.
- With respect to *representing constraints*, in both the EMS and the IH tasks the designer concentrated on developing constraints whenever his model of the object being designed was under-constrained. In the EMS this occurred in the early part of the design, as the designer attempted to develop a mental model from the design specifications. Here, due to the designer's experience with the task, constraining the original problem statement enough to develop a model of it was not a problem. Instead the development of constraints occurred as he worked on the chip, the one aspect of the task which was least familiar.
- It is interesting to note that in this situation the use of plans was frequent, and replaced the need for *both* simulation and note making.

In summary, we saw in the IH that when the designer is familiar with the particular object being designed he tends to rely more on previously developed plans and less on the powerful but attention-consuming tools of simulation and note making.

IV. Comparison of Novice and Expert Behavior

Here we compare each of the behaviors found in the experts to the behavior of the novices. The comparison will highlight the role that experience plays in the expert's knowledge and skill in software design.

1) Behavior—Formation on Mental Models: The novices represented commands only as pseudocode. In Section III-A, we defined mental models as representations capable of supporting, simulation. As discussed below, the pseudocode of the novices was not used for simulation. As a result, we prefer to think of the novices as having representations, rather than models.

Both of our novice subjects always considered only one EMS command at a time; This is very different from the experts' treatment of the EMS, in which each command was considered in the context of all of the potential interactions of the system.

2) Behavior—Systematic Expansion of Mental Models: As stated above, the novices seemed only able to represent commands at a single level, the level of the pseudocode. This is illustrated by the following quote: At 3 minutes into the task, novice N1 said:

[4]It seemed as if he were trying to find the novel part of the task and explore it in order to make the task more challenging. This desire to be challenged sems an interesting part of the expert's way of solving problems; it certainly is adaptive in that it makes the expert better with every new problem he solves.

"To save I have to open a file and then write to that file...
If I have 5 or 6 messages I have to consider if I want to save all of them or whether I should save all of them or whether I should save a specific one and specify which one I am saving."

While making these statements N1 wrote the following pseudocode:

SAVE
OPEN FILE
READ FILE

In comparison to the above, at 3 minutes into the EMS task, expert E3 said:

'I guess I have to establish a set of assumptions of my own"

A comparison of the above two quotes points out the difference between the problem-expansion methods of the novices and the experts. For the experts the expansion is divided into several levels. For the novices only *one* level of representation is possible.

3) Behavior—Simulation of Mental Models: Once again we find a sharp contrast between the novices and the experts. The experts continually and explicitly conducted mental simulations at different levels of concreteness. As mentioned above the novices did not seem to have a simulatable model.

N1 had one persistent problem that would have been caught had she been able to represent her commands in a way that supported simulation. The problem resulted from N1 having conflated two meanings of READ, i.e., having the user read a message and having the operating system read a command from the user to the mail system. Simulation would have allowed this bug to be caught because in a *global* simulation the designer would have simulated *all* of the results of READ and so the two senses of the term would have been noticed sooner. Without simulation, N1 tried to have her READ command fulfill the functions implied by both senses of READ and ended up trying to design something which would put messages up on a terminal screen and, as a result, get an input from the user, telling the EMS what to do next. At the end of the design session N1 realized that the "read input from the user" sense was not the one she wanted. She than was able to successfully work out the "let the user see a message" sense of READ.

4) Behavior—Representing Constraints: This behavior was not found in either of our novices. However, the sum total of the novices knowledge about mail systems is quite impoverished. Therefore it is reasonable that it would not be able to support the kind of inferencing that the expert knowledge bases supported.

5) Behavior—Retrieving Labels for Plans: The novices repeatedly represented EMS commands only as pseudocode. These lines of pseudocode seemed to be the novices *plans* for the EMS commands. When N1 said "to SAVE I have to open a file" and simultaneously wrote "OPEN FILE" we had the sense that the pseudocode was her plan for SAVE. We can compare the low-level plans of the novices to the high-level plans used by the experts. For example, in the EMS task E1 had a plan in which messages were represented simply as data objects; he did not need to specify any of their implementation details. This novice–expert difference seems similar to findings by Adelson, that novices have only low-level representations of programming knowledge, whereas experts have representations at both abstract and concrete levels [1].

6) Behavior—Note Making: Both N1 and N2 did, at one point make notes for themselves. However, these notes *only* made while considering the problem statement we had given them. The following quote from N2 illustrates this point.

"For READ...
[Pointing to READ on the sheet where the problem statement was written]
I have to put the mail box in a good directory...
I have to find somewhere to put the user's mailbox...
So I'll have to study the operating system and the file system."

The above note making is a contrast to the more conceptually driven notes made by the experts, which were usually formed while considering the properties of the command they were currently considering.

A novice–expert difference appeared not only in what prompted the designers to make a note, but also in the likelihood of resolving the note once it was made. Whereas our experts usually did return to their notes at the appropriate time, N1 attended to the note almost immediately whereas N2 did not return to the above described note at all. However, since the novices are not following a systematic expansion they feel free to take care of their notes immediately (or not at all).

In summary, the novices behavior was dominated by the use of pseudocode which seemed to be their only way of representing the commands of the EMS. This representation did not support simulation, allow them to develop constraints, or follow systematic expansion with its associated note making.

V. Summary and Conclusions

In this paper we have discussed the functionality of the following behaviors:

1) Formation of Mental Models: Mental models are formed to allow the designer to simulate his design-in-progress.

2) Simulation: Simulation allows the designer to integrate familiar material in novel ways. It does so because it allows the designer to examine all of the unforeseen interactions in a novel assembly of well-known functions. Simulation also allows the designer to check the behavior of the partially specified mechanisms in his design-in-progress against the design's goal state behavior.

3) Systematic Expansion: Systematic expansion allows simulation to function smoothly by ensuring that the input to all elements are at the same level of detail.

4) Representing Constraints: Designers need to make the implicit constraints on a design explicit when they are buiding a mental model that is unfamiliar to them and therefore insufficiently constrained for simulation.

5) Retrieving Labels for Plans: When designers encounter a part of the problem that they have solved previously they simply use a label to indicate that they have a plan stored in memory which can be retrieved and used in the design.

6) Note Making: Note making allows the designer to maintain his systematic expansion, without overlooking important concerns which are not at the current level of detail. The notes serve to remind the designer of the concern at the appropriate time.

Additionally we have shown that these behaviors are affected by the designer's experience with the object being designed and the domain being designed in:

- Simulation and note making are only found when the designer has sufficient domain knowledge.
- When the designer does not have sufficient experience with the object being designed he will develop the constraints on the design in order to get sufficient specificity for the simulation process.
- When, as a result of experience, the designer has an appropriate plan he will use the plan rather than formulating constraints, simulating, and making notes.

The implication for research on tools to support software designers is that designers can be aided by a whole range of tools. When a tool will be helpful will depend on the level of the designer's experience with the object being designed and the domain being designed in. For example, when the designer has designed an object previously, he will need tools that help retrieve the previous designs. When a designer has experience with elements of the design but not with the design as a whole, he needs tools that will help to retrieve and assemble the elements in a simulation. When the designer has not had experience in the domain he will need tools that help him to infer the constraints of the design.

References

[1] B. Adelson, "The development of abstract categories in programming languages," *Memory & Cogn.*, vol. 9, no. 4, pp. 422–433, Aug. 1981.
[2] J. Anderson and R. Jeffries, "Errors due to losses from working memory," Tech. Rep. CMU-P S-84-145, 1984.
[3] W. Chase and H. A. Simon, "Perception in chess," *Cogn. Psychol.*, vol. 4, pp. 55–81, 1983.
[4] M. Chi, R. Glaser, and P. Rees, "Expertise in problem solving," in *Advances in the Psychology of Human Intelligence*, vol. 1. Hillsdale, NJ: Erlbaum, 1981.
[5] A. Collins and D. Gentner, "Simulations of mental models," in *Proc. Cogn. Sci. Soc.*, 1982, pp. 185–189.
[6] J. DeKleer and J. S. Brown, "Assumptions and ambiguities in mechanistic mental models." in *Cognitive Skills and Their Acquisition*, Anderson, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1981.
[7] K. A. Ericsson and H. A. Simon, "Verbal reports as data," *Psychol. Rev.*, vol. 87, no. 3, pp. 214–51, May 1983.
[8] R. Jeffries, A. Turner, P. Polson, and M. Atwood, "The processes involved in designing software," in *Cognitive Skills and Their Acquisition*, Anderson, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1981.
[9] E. Kant and A. Newell, "Problem solving techniques for the design of algorithms," Tech. Rep. CMU-C S-82-145, 1982.
[10] D. Littman, J. Pinto, B. Adelson, and E. Soloway, in preparation.
[11] A. Newell and H. A. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
[12] E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem, "Basic natural categories," *Cogn. Psychol.*, vol. 8, pp. 382–439, 1976.

Beth Adelson received the Master's and Ph.D. degrees from Harvard University, Cambridge, MA, in 1981 and 1983, respectively.

She is currently a Research Associate in the Yale Artificial Intelligence Laboratory, New Haven, CT, and is also Associate Program Director for the National Science Foundation Division of Information Science and Technology for the academic year 1985–1986. Her research interests focus on models of problem solving in artificial intelligence and cognitive science.

Elliot Soloway received the B.A. degree in philosophy from the Ohio State University, Columbus, in 1966, and the M.S. and Ph.D. degrees in computer and information science from the University of Massachusetts, Amherst, in 1973 and 1978, respectively.

Since 1981, he has been a member of the faculty at Yale University, New Haven, CT, where he is currently an Associate Professor in the Department of Computer Science. He and his group are engaged in studying the cognitive underpinnings of various aspects of programming: program design, generation, comprehension, debugging, and maintenance. As a complement to this theorizing, he and his group develop AI-based software tools that specifically address areas in which experts and novices need assistance. He is also currently Vice President of Compu-Teach, Inc., a company in New Haven that produces educational software.