CHAPTER 5

# Breakdowns and Processes During the Early Activities of Software Design by Professionals

**Raymonde Guindon**
**Herb Krasner**
**Bill Curtis**
**Microelectronics and Computer Technology Corporation**
**9390 Research Boulevard**
**Austin, Texas 78759**

## ABSTRACT

This chapter summarizes some of the main breakdowns (or difficulties) occurring early in the software design process when professional designers work on a problem of realistic complexity. One class of breakdowns is caused by lack of knowledge and another class is caused by cognitive limitations. A third class of breakdowns is caused by a combination of these two factors. The main breakdowns observed are: 1) lack of specialized design schemas; 2) lack of a meta-schema about the design process leading to poor allocation of resources to the various design activities; 3) poor prioritization of issues leading to poor selection of alternative solutions; 4) difficulty in considering all the stated or inferred constraints in defining a solution; 5) difficulty in performing mental simulations with many steps or test cases; 6) difficulty in keeping track and returning to subproblems whose solution has been postponed; and 7) difficulty in expanding or merging solutions from individual subproblems to form a complete solution. We have also observed *serendipitous design* and the process of understanding and elaborating the requirements through exploration of the designer's mental model of the problem environment. This study provides many observations of breakdowns and design behaviors not reported in previous studies and necessary prior to developing a model of the cognitive activities during software design. This study also provides critical information to guide the design of tools and methodologies to improve the efficiency of software designers.

## MOTIVATION AND GOALS

The goal of this study is to identify the breakdowns most often experienced by professional software designers and determine the software tools that would alleviate these breakdowns. This chapter will not describe these tools but will concentrate on describing the breakdowns. Breakdowns have been broadly defined as ineffective design activities, undesirable consequences of these ineffective activities, activities that are difficult to perform because they tax the designers' limited cognitive resources, or causes of these ineffective or difficult design activities. These breakdowns are likely to produce design solutions that are incorrect. Our strategy toward the identification of breakdowns has been to:

- Study designers with real, extensive, and widely varied software design experience.
- Give the designers a more complex and realistic problem than has been given in other studies of software design, yet not so different that our results cannot be easily compared to them (e.g., Jeffries, Turner, Polson, and Atwood (1); Adelson and Soloway (2); Kant and Newell (3)).
- Use the observational technique of thinking aloud protocol rather than controlled experimental manipulations because of the scarcity of previous empirical work on software design. This initial study is exploratory and we have observed many design behaviors and many breakdowns that will provide a foundation for modeling cognitive activities during software design.

Our study differs from other studies of software design by individuals on two main points. First, we are taking a next step in advancing the empirical study of software design by using a more complex and realistic design problem than used in other software design studies. Second, this study is especially oriented toward identifying the breakdowns occurring during software design by professional designers.

## DESCRIPTION OF THE METHODOLOGY

### Participants

Thinking aloud protocols were collected from 8 professional programmers and system designers. Three experienced designers were selected for a full protocol analysis from these eight professionals. P6 had a Ph.D. in Electrical Engineering with more than 10 years of professional experience, mainly in communication systems and hardware architecture. P8 had a Masters in Software Engineering with 5 years of experience, mainly in real time systems. P3 was a Ph.D. Candidate in Computer Science with 3 years of professional experience, mainly in logic programming. These particular designers were selected because they were considered by their peers to be experienced and competent designers, because of the wide variety of their educational backgrounds and years of experience, because their design solutions were considered the best among the eight designers, and because of the wide variety of their design strategies and solutions. We deliberately choose to analyze the widest spectrum of design behaviors in order to gather a wide variety of breakdowns and design strategies.

### Problem Statement

The *lift control problem* is currently a standard problem used in the areas of software specification and software requirements modeling research. The goal is to design the logic to move N lifts between M floors given the constraints expressed in the problem statement. The problem statement is given in the Appendix.

### Procedure

Thinking aloud reports were collected from participants who were asked to design the logic for the N-lift problem. They were given two hours to produce a design solution that was in such a form and level of detail that it could be handed off to a competent system programmer to implement. The participants were videotaped and given paper and pencil to work their solution. The notes and diagrams produced by the participants were time-stamped regularly by the experimenter. The transcript of each participant was also time-stamped, and the written notes and diagrams were included in the transcript. The procedure departed from typical verbal protocols in that the experimenter intervened more substantively, often acting as a client and sometimes providing some help when the participants encountered difficulties. There were two reasons for this departure. First, in realistic design situations one or more people are typically available to answer

questions or arbitrate issues in the requirements. This communication and negotiation between client and designer is perceived as a critical element in early design. Carroll, Thomas, and Malhotra (4) have provided an analysis of the cyclic nature of these interactions. At least one participant, P6, would frequently want to discuss design issues with the experimenter as this represented his normal mode of designer-client interactions. However, such a veridical feature conflicts with the traditional methods for collecting and analyzing verbal protocols. Second, our objective was to generate as much design behavior as possible in order to observe a broad range of design breakdowns. The participants were selected and the verbal protocols collected by the second author. The videotapes and transcripts were independently analyzed by the first author later.

### Protocol Analysis Process

The process of protocol analysis was divided into three major steps:

1. Enumeration of possible cognitive activities that could occur during the session based on previous studies, the problem-solving literature in psychology, and artificial intelligence models of design. The function of these preliminary models is to guide the protocol analysis, though not limit it. New activities or interactions between them are also sought.
2. Segmentation of the protocols into episodes indicating breakdowns or corresponding to cognitive activities during software design.
3. Identification of the relations between the software design activities. Four main types of relations were identified: 1) temporal (e.g., precedence, iteration, interruptions, and resumptions); 2) transition between internal, mental activities and external activities (e.g., from mental to external representation of lifts and floors on paper); 3) transition between activities dealing mainly with the problem domain (e.g., lifts, floors, buttons) and activities dealing mainly with the solution domain (e.g., control structures, data structures); 4) functional composition (e.g., the activity of understanding the requirements was composed of shorter episodes such as disambiguation of the problem statement through mental simulations, the addition of an assumption, or the abstraction of critical statements).

Verbal protocol analysis is essentially an exploratory observational technique particularly suited for research in new domains. The study of cognitive processes during software design is such a domain.

### Issues of Validity Generalization

There are three issues that must be addressed in determining the validity of generalizing results from these data to realistic design situations. These issues deal with the task, the sampling of participants, and the external validity of the experimental situation. As stated earlier, the lift problem is a standard exercise in research on software specification techniques. Although it is not as complex a task as, for example, designing a distributed electronic fund transfer system, it is nevertheless a next step in increasing the technical challenge offered by tasks used in empirical studies of design. Thus, as we begin to understand how designers marshal their cognitive resources to solve problems of this complexity, we can move on to tasks of even greater complexity.

The second issue concerns how representative our sample of participants is of the larger population of software designers. There is simply no reliable way to answer this question given the current maturity of the field. That is, there are no population data available against which to compare the variability of our sample. There is no standard type of individual who becomes a software designer. Their educational backgrounds,

work experience, job settings, and variability of skills differ radically. Furthermore, there is not even agreement on the relevant variables to measure if we wanted to characterize this population. Another significant problem is that there is no standard job title or description for those who perform software design.

We attempted to get a broad sampling of educational backgrounds, application experiences, and previous working environments in the eight designers selected to participate. Further, we selected three protocols for analysis that represented completely different strategies in attacking the design. We make no claim that these protocols represent the full range of strategies in the larger population of software designers. Rather, we believe that since it is unlikely that a multi-environment population study of software designers will be funded in the foreseeable future, the description of this population must be pieced together from studies like this that describe the problem-solving characteristics of a few designers in great depth.

The third issue concerns the extent to which the conditions under which the data were collected are representative of those under which actual design occurs. Clearly, designing programs of any significance normally takes more than two hours, unless the designer has extensive experience in designing programs of great similarity. Our goal in collecting these data was to gather information on a concentrated problem-solving effort that would provide a broad range of cognitive behaviors and would display an interesting array of breakdowns. We cannot be sure that additional breakdowns would not have been observed had we carried the data collection out over several days or even weeks with a more complex problem. Furthermore, some of the breakdowns may have been exacerbated by the concentrated nature of the two-hour session. However, other studies of software design by individuals have collected verbal protocols over a session of two hours (Jeffries, Turner, Polson, and Atwood (1); Adelson and Soloway (2); Kant and Newell (3)). Therefore, we can at least compare our results to their results legitimately.

Finally, some of the breakdowns and processes we have observed were also reported by Kant and Newell (3) and Adelson and Soloway (2), and more importantly in a study of mechanical engineering design using a ten-hour design session by Ullman, Stauffer, and Dietterich (5). This overlap between some of our findings and findings in other studies performed in other domains and under different conditions supports the validity and generalizability of our new findings.

## GENERAL OBSERVATIONS ON DESIGN BEHAVIOR

The three designers adopted very different strategies during design. A brief characterization of their overall strategies follows.

P6 seems to have the most relevant specialized computer science knowledge to solve the N-lift problem - he has **specialized design schemas** relevant to distributed systems. P6 uses these design schemas to **decompose the problem into simpler subproblems**. He opts for a distributed control solution. He then decomposes the problem into two subproblems, communication between each lift and route scheduling by each lift. However, he clearly has better design schemas for the communication subproblem than for the scheduling subproblem. He successively refines his solution for the communication subproblem while he performs much more exploratory design for the scheduling subproblem. By exploratory design, we mean design with many mental simulations of the problem environment and mental simulations of tentative solutions unguided by a plan. Another aspect of his design activities is that they are **issue-driven**: he identifies some critical features his solution should have, such as

*reliability and no single point of failure*, and uses them to **control** the selection of alternative solutions for many of the subproblems (e.g., selection of a control structure and selection of a communication scheme between lifts). Moreover, P6 also consciously generates many **simplifying assumptions**, which he evaluates for their plausibility, as a **complexity reduction strategy**. So, P6's process is mainly characterized by the use of specialized design schemas, by being issue-driven, and by the generation of simplifying assumptions.

P8 seems to follow more closely than the other participants a **meta-schema for design**. This meta-schema seems to be derived from software engineering practices, especially the Jackson System Development method (6). P8 explicitly acknowledges the need for **exploring the problem environment** to achieve a good understanding of the requirements before seeking a solution. The problem environment is the set of objects, events, and behaviors in the domain relevant to the computer system being designed (e.g., floors, lifts, buttons, buildings, people waiting, fires). An important part of his design activities is the representation of entities and relations in the problem environment and their mental simulations. However, P8 does not seem to have as relevant specialized design schemas as P6, which may have induced him in exploring the problem environment. Possibly as a consequence of exploring the problem environment, the design process of P8 is partly controlled by recognition of **partial solutions, at different levels of detail or abstraction**, without having previously decomposed the problem into subproblems. We call such design process **serendipitous design**. As another aspect of P8's process, he decomposes the problem into handling the requests coming from floors and handling the requests coming from inside the lifts. He also attempts, within this problem decomposition, to solve initially for one lift and then attempts to expand the solution to N lifts. However, he experiences great difficulty in merging the partial solutions from handling requests from floors and requests from lifts and expanding the solution from 1 lift to N lifts. The decomposition of the problem into requests from floors and requests from lifts, and the reduction of the problem from N lifts to 1 lift does not appear to be based on a specialized design schema. Such a design schema would have provided P8 a plan for a well-motivated decomposition of the problem into subproblems and a merging of the partial solutions. So, P8's design process is characterized by the use of a meta-schema for design, by exploration of the problem environment, by serendipitous design, and by difficulty in merging the partial solutions.

P3's design activities appear the least systematic and the most locally governed. They are the least systematic because P3 does not seem to use a meta-schema for design and he does not seem to use specialized design schemas to guide the decomposition of the problem into subproblems. They are the most locally governed in the sense that he does not exhibit exploration of the problem environment and consideration of alternative solutions to problems. P3 does not, as opposed to P6, consider one or more issues as crucial and select the solution from a set of alternative solutions that best satisfies the issues. His approach seems mostly governed by a **familiar computational paradigm**, logic programming. He produces many cycles of generating a tentative solution, simulating it, debugging it locally, and simulating it again. He has difficulty with mental simulations of the many tentative solutions and with keeping track of the test cases during these simulations. So, P3's design process is characterized by a generate-test-debug strategy and difficulty with mental simulations of tentative solutions.

Likewise, the solutions produced by the designers are quite different. We will discuss mainly the solution architectures and the representation schemes of the solutions.

P6's solution is a communicating ring of independent elevators. He selects

distributed control over centralized control. In this scheme, each elevator operates on its own and passes information around the ring to the others, coordinating the schedules of pickups that they had decided on independently. P6's solution is represented as finite state machines. In each elevator there are two communicating finite state machines; one for handling the local processing of the elevator as it decides what stops to make along its route, and the second for communicating with the other elevators independently.

P8 adopts the classic star architecture in which the elevators communicate through a central process. P8 uses abstract data types, data flow diagrams, and pseudocode as representation schemes.

P3 works at developing a global model of system behavior described as a set of logical assertions with initial thoughts of centralized control. P3 represents the behavior of the system by logical assertions written in a Prolog style.

So, experienced designers can exhibit a very wide variety of design strategies, both between and within designers. This variety of design strategies is also accompanied by different types of breakdowns (which will be described in the next section). Finally, design solutions vary widely between designers. These observations highlight the complexity of the design process. These observations also highlight the critical contribution of specialized knowledge to the design process, and as a consequence, the wide individual differences that will be observed between designers' strategies. Finally, these observations indicate that a wide variety of tools and methodologies are needed to best support the variety of design strategies.

## SOME OF THE BREAKDOWNS OBSERVED DURING DESIGN

The breakdowns that we are reporting consist of both symptoms and causes of difficulties during the design process. Moreover, these breakdowns are not necessarily independent of each other. We observed two main classes of breakdowns, with a third class being a combination of the other two. The first class consists of **knowledge-related** breakdowns. They are due to 1) lack of specialized knowledge of computational solutions corresponding to characteristics of the application domain, 2) lack of knowledge/experience of the design process itself, or 3) lack of domain knowledge. The second class of breakdowns are due to general **cognitive limitations**. They result from 1) capacity limitations of short-term or working memory, and 2) the unreliable retrieval of relevant information from long-term memory. They are also due to the weakness of our standard tools and methodologies aimed at alleviating cognitive limitations (e.g., checklists). The third class of breakdowns results from a combination of both knowledge deficiencies and cognitive limitations. These latter breakdowns occur because the lack of relevant specialized knowledge induces designers to use weak problem-solving methods, such as generate and test and means-end analysis. Unfortunately, weak methods can be very taxing cognitively and they often translate into poor performance because they require search of a large space of possibilities (Laird, Rosenbloom, and Newell (7)). Moreover, the lack of specialized knowledge is also associated with a lack of cognitive knowledge structures supporting memory for the activities during the design process (e.g., supporting memory for postponed subproblems or memory for test cases).

## Knowledge-Related Breakdowns

**Lack of Relevant Problem-Specific Design Schemas.** The main determinant of performance appears to be the presence or absence of specialized design schemas. Design schemas are mental representations of software design families. Borrowing from a definition given by Rich and Waters (8), a design schema consists of a set of roles embedded in an underlying matrix. The roles of the schema are the parts that vary from one use of the design schema to the next. The matrix of the schema contains both fixed elements of structure (parts that are present in every occurrence) and constraints. Constraints are used both to check that the parts that fill the roles in a particular occurrence are consistent, and also to compute parts to fill empty roles in a partially specified occurrence. These schemas can be instantiated through refinements and specializations to particular instances of software designs. The design schemas embody the knowledge of alternative solutions to classes of problems. Problem-solving using design schemas proceeds through recognition that the requirements may be an instance of a known design schema followed by propagation of constraints from the explicit and implicit requirements and specialization of the design schema. Examples of design schemas and their specializations are given in the work by Lubars and Harandi (9) and Rich and Waters (8). The Inventory Control System design schema can be specialized into the Reservable Inventory Control System schema and the Non-Reservable Inventory Control System schema. A Library Inventory Control System schema is a specialization of the Non-Reservable Inventory Control System schema. In the Library Inventory Control System schema the role Check Out Book (an operation) is a specialization of the role Dispense Inventory specified in the Non-Reservable Inventory Control System schema.

Regarding retrievability, a design schema is composed of a description of the conditions under which its solution is relevant. These conditions contain an abstract representation of critical features in the given problem environment. For example, in the N-lift problem the abstract representation could be in terms of many clients who might make simultaneous requests for service to many servers (lifts) at different times and locations. Such a description could be sufficient to retrieve a design schema appropriate for the N-lift problem. Regarding problem decomposition, a design schema also contains a solution plan to guide the decomposition of the problem into subproblems, each subproblem with its own design schema. The design schema as a cognitive structure also supports the storage and retrieval of intermediate solutions and backtracking if necessary, and as a consequence, reduces working memory load during design and increases the probability that partial solutions and postponed subproblems will be retrieved when needed. The design schema also guides the expansion of a reduced solution or the merging of individual solutions to subproblems into a complete solution.

P6, because of his specialized design schemas for distributed and communication systems, could quickly identify the main design issues (i.e., no single point of failure) and knew alternative solutions to be evaluated (e.g., central vs. distributed solutions for control and communication between lifts). He could also quickly retrieve from memory, for example, known solutions for posting messages and for avoiding a race condition. However, for the subproblem of scheduling service and especially its subproblem of choosing a route, for which he seems to lack design schemas, his design process appeared much more exploratory and accompanied with mental simulations.

The following excerpts from the protocols indicate the use of such schematic knowledge. P6 immediately recognizes that in scheduling N lifts the problem of control arises. His design schema for control represents two alternatives, centralized and

distributed, which he immediately retrieves. His knowledge of the alternatives also contains their advantages and disadvantages. He opts for distributed control. He then recognizes the next problem to solve, communication between lifts. He retrieves the possible alternatives and again opts for distributed communication between lifts, where each lift broadcasts which requests it will service to all other lifts. He then recognizes the problem of a race condition and immediately retrieves from his design schemas a solution to the race condition problem, that is, arrange the lift in a ring for sequential polling of the lift requests.

P6- *"I'm going to schedule the elevators. Do we have a central controller? It doesn't say in the problem. We have a central controller or a distributed controller, is that up to me? ... The good news about central control is it's an easier algorithm vs. distributed control. The bad news is that you have a single point of failure... I'll start off by thinking about a distributed control system..."*

P6- *"Let's say all the elevators are sitting down at the bottom floor and there's an up button pressed on floor three. Someone's got to post that request and someone's got to pick it up and mark it that he's responding to it. That implies a centralized place to post the requests and we're back to the single point of failure... Well, you could broadcast a message to all the other elevators that you are servicing number one. Seems like the algorithm is: every elevators look at all the buttons all the time; he sees an up posted on floor number three; ... one of them grabs it and post a message ..."*

P6- *"Now we have to make sure we don't get in a race condition. And we can do that algorithmically usually. You can do some kind of ring system by allowing them to scan in sequence and tell one another when they've got scanned so they don't get in a race."*

P3 applies a familiar computational paradigm, logic programming, but seems to lack relevant specialized design schemas. P3 recognizes at the beginning of the session that the lift problem is not a type of problem he is used to solve: *"... this is different from what I am used to thinking about because we don't just have one lift we can decide algorithmically what to do about ..."*. As a consequence, P3 adopts central control for its computational simplicity: *"... Maybe I can assume there's some central processor that can receive signals from all the lifts and decide on things? ... Well, I think it might be easiest to do it that way."*.

**Lack of or Poor Meta-Schema for Design.** The next main determinant of performance appears to be the presence or absence of a meta-schema of software design. A design meta-schema is a schema about the process of design itself and not about a particular class of problems. The meta-schema is used to **control** the design process. A design meta-schema guides the execution of design activities and resource management. A design meta-schema represents design process goals and their alternatives and guides the amount of effort spent in different activities. For example, a design meta-schema will help answer questions such as:

- How much time and money should be spent on the complete design process?
- How much time and money should be spent on exploring the problem environment?
- How much time and money should be spent exploring alternative solutions for identified subproblems?
- Which subproblem should be attempted to be solved next?
- How many alternative solutions should be considered for the selected subproblem?

P8 has been trained as a software engineer and he seems to be using a meta-schema about design to guide the resources he allocates to various design activities and the amount of exploration done. This meta-schema is based on the design technique developed by Jackson (6). P8 first explores his mental model of the problem environment using various representation techniques to a much greater extent than P6 and P3 before proceeding to define an initial solution. The *problem environment* is a part of the real world, outside of the designed computer system, with which the computer system interacts. It contains the objects, properties, behaviors, and constraints in the world that are relevant to the design of a solution. In our case, the problem environment of a lift system contains such entities as floors, lifts, floor buttons, lift buttons, people, people waiting, requests for lifts, buildings, lift doors, lift panels, the safety of passengers, etc. The designer produces a mental model of the problem environment, possibly incomplete and inconsistent, based on the requirements and the designer's knowledge about the world. The mental model of the problem environment specifies more information than is contained explicitly in the requirements, and will often be more complete than the requirements or may sometimes be inconsistent with the stated requirements. The purpose of exploring the problem environment is to increase the completeness of the requirements and to discover unstated constraints, properties, behaviors, or objects that may generate constraints on the solution.

The following excerpt exemplifies how P8 uses a meta-schema about design to guide how much time he spends on various design activities and the amount of exploration done. About twenty minutes after the beginning of the session, while P8 is still exploring the problem environment, he shifts to a subproblem at a very low level of detail. He starts exploring the subproblems of handling inputs on an interrupt basis and of scheduling the service to lift requests. He then dramatically shifts back to a high level of abstraction. *"But at this point I feel I might be getting ahead of myself, so I want to think about other basic strategies. For instance, usually when I'm doing a design I try to think about things in a data abstraction way or object-oriented way. So now I'm trying to see if I can think about the objects and the system kind of independently."* About 5 minutes later, one can again see from the protocol how P8 seems to apply conscious strategies of resource allocation: *"... I'm sort of working at a high-level now, but I just have the feeling that if I maybe just tried working at a lower level for a second I might get some ideas ... just kind of imagine in my mind how I could actually do this at some deeper* [i.e., more concrete, precise] *level"*. Again about 14 minutes later, P8 makes clear that he is applying some strategies to allocate resources to different activities during design: *"... I feel I still need to think about the problem. On the other hand, I feel a little bit frustrated because I don't know why things haven't gelled enough yet. And so on the other hand, I'm impatient to sort of have things gelling; but on the other hand, I feel like there's some relationship between lift requests and floor requests that I feel I just haven't really grasped that yet."* P8 recognizes that he should explore the problem in greater detail before adopting a solution.

P6 combines exploration of the problem environment and issue-driven design. P6 does not seem to use as sophisticated a meta-schema about design as P8. Nevertheless, about sixteen minutes after the beginning of the session, where he has already explored cursorily the subproblems of posting lift requests and servicing the requests, his use of a meta-schema based on software engineering practices is revealed by his comment: *"I'm just putting down the criteria which I'm basing this algorithm on, and I want to be sure I get the requirements that I'm trying to satisfy before I get the solution... I do not want to fall in the trap of solutions without requirements."*

We hypothesize that the use of a meta-schema for design is particularly useful if

the designer lacks more specialized relevant design schemas. The use of a meta-schema about design helps the designer control the amount of effort spent in different activities during design (e.g., exploration for understanding the problem environment, exploration and evaluation of different solutions).

**Poor Prioritization of Issues Leading to Poor Selection from Alternative Solutions.** Tong (10) views design as a dialectic between the designer and what is possible. Design can be conceived as the process of producing an optimized artifact given a set of interrelated or independent constraints, explicit or implicit, imposed by the problem, the medium, and the designer (see also Mostow (11)). Examples of constraints provided by each of these sources are:

- the problem
  - a given (perhaps informal) functional specification
  - limitations of the available media (e.g., available hardware and software)
  - implicit and explicit requirements on performance and usage (e.g., cost, power, speed, space)
  - implicit and explicit requirements on the form of the artifact (e.g., maintainability, reliability, reusability, simplicity)
- the design process itself
  - time available
  - allowable costs
  - tools available (e.g., type of workstation, special hardware)
  - team work or individual work
  - organizational procedures
- the designer
  - knowledge of the application domain
  - knowledge of the class of system being design (i.e., specialized design schemas)
  - knowledge or experience of the design process itself (i.e., meta-schema for design)
  - cognitive and motivational attributes.

Very good designers seem to know how to prioritize and balance these constraints on the basis of their domain knowledge, knowledge of the type of system to be designed and developed (i.e., design schemas), and their knowledge of the design process itself (i.e., design meta-schema). Following this prioritization, they allocate their time to the various subproblems according to their relation to these priorities. So, this third breakdown is related to management of resources and to the meta-schema of the design process.

Part of this prioritization process is the evaluation of alternative solutions based on a set of selected criteria as described above. These evaluation criteria are called *issues and their alternatives*. For example, P6's main issue is reliability and no single point of failure. He infers this issue from his knowledge of the problem domain and the class of system he is designing. When dealing with the subproblem of control, and its alternatives as centralized vs. distributed, he evaluates the two alternatives and opts for distributed control because of its perceived greater reliability. He does likewise when considering alternatives for communication schemes between lifts. The following excerpt demonstrates his evaluation of alternative solutions on the basis of the inferred issue of no single point of failure.

P6- *"The good news about central control is it's an easier algorithm vs. distributed control. The bad news is you have a single point of failure... You would rather not have a single point of failure because if, you know if all the elevators go*

*down because it goes down... You would not want everything to go down... So, I'll start off thinking about a distributed control system .... you got to post that request and then someone's got to pick it up and mark that he's responding to it. That set of words implies a centralized place to post the request and we're back to the single point of failure. ... Well, you could broadcast a message to all the other elevators that you are servicing number one. Seems like a simple algorithm ..."*

**Breakdowns Due to Cognitive Limitations**

**Difficulty in considering all the stated or inferred constraints in refining a solution.** This breakdown represents a failure to integrate known or assumed constraints in the design solution. These failures occurred even though these constraints were explicitly given in the problem statement and the problem statement was available throughout the session to the designers.

In the example below, P3 knows the constraints that all floors must be serviced equally and that direction of travel must be kept. In his design he decides to keep the elevator going in one direction until all outstanding requests in this direction are satisfied.

P3- *"If there are still requests and those requests are higher than we want to go, then we'll keep going up... If an elevator was going up, it keeps going up until all up requests are satisfied ... that insures that requests will eventually be met, otherwise there could just be oscillating between floor one and floor two."*

Nevertheless, later, he forgets about his solution on these constraints and evaluates improperly his overall solution. P3 seems to believe that requests to go down while the lift is going up are going to be serviced immediately. As a consequence, the lift would not keep direction of travel, violating a problem constraint.

P3- *"well this is interesting now, it brings up the questions of what if an elevator is on its way up to a floor to pick up somebody up there and a person snatches it at the floor below and wants it to go down... it's going to be priority driven and that's nasty..."*

A somewhat related breakdown is to disregard certain relevant aspects of the problem - the "rose-colored glasses" syndrome. In the excerpt below, P3 proposes a non-deterministic solution to allocate floor requests to lifts. He also acknowledges that this may jeopardize the requirement that all floors be given equal priority, thereby violating a given constraint. He nevertheless adopts the non-deterministic solution without any more analysis of its consequences.

P3 - *"Our processor will look to see what elevators are moving up and down. We'll do this non-deterministically. Maybe that's dangerous for this priority* (all floors given equal priority) *but OK."*

**Difficulty in performing complex mental simulations with many steps or with many test cases.** Designers find it very difficult to mentally simulate their partial or complete solutions. They find difficult to simulate the interactions between components of the artifact. They also find difficult to simulate the behavior of a component if it extends over many steps. We also observed the multiple simulations of the same test case and the failure to simulate a crucial test case during evaluative simulations, leading to incorrect assessment of the correctness of the solution to a given subproblem and hindering progress on other subproblems. To help mental simulations, designers often resorted to diagrams. However, because they were poor medium to

represent changes in location and time, they were not sufficient to prevent the breakdowns. The following excerpts show how designers had difficulty even with the simplest simulations.

P8 - *"... it's kind of confusing, there's lifts* (requests) *and there's floors* (requests) *and it says "all requests for floors within lifts must be serviced eventually with floors being serviced sequentially* (in the direction of travel). *Apparently that means ... Let me give a better example ... I'll have to draw a picture."*

P3 - *"...the way I've written this doesn't capture the continuity of direction. 'Cause this just looks to see some other. We've happened to have reached some floor, this now just picks some other request. Oh, no, that's right, this tries to find a request that's in the same direction."*

P8 - *"When an interrupt happens it adds a request to my list and the structure of requests are floor request, a lift request, or the emergency button... Let's say the third guy wants to go the fifth floor. Let's say there's a floor request on the. Oh, I missed something here. Floor request has originating floor and direction... Could I borrow that pencil?* (to draw a picture)."

The next excerpt shows how the mental simulation of the test cases is at the beginning quite systematic and quickly becomes unsystematic and finally incomplete as other concerns attract the attention of the designer.

P3 - *"... Given the fact that somebody in a lift has pushed a 'go to floor' button (that's a floor he wants to go to). First of all, if that lift is at that floor then I can just de-illuminate the button..."* Here, he digresses for about one minute. *"... On the other hand, if he push a lift button and he's not at the floor he requested, what I'll do is I'll put that into this global base which means that by the deamon watching, the light will go on. And that's all I really have to do other than examine this and decide which way the elevators move. So I'm trying to handle the request right now... So really all I'm doing is filtering out requests that can be handled immediately, but OK, let's go with that for a second.... I'll handle emergencies later."* He digresses for about 30 seconds and then changes topic drastically. He abandons the simulations of the other test cases, those requests that cannot be handled immediately. *"Ok. Now comes time to build a mechanism which causes all the stuff to change."*

### Breakdowns Due to Lack of Knowledge and Due to Cognitive Limitations

**Difficulty in keeping track and returning to aspects of problems whose solution refinements have been postponed.** When focusing on one aspect of the problem and postponing the solution refinement of others, the designer must be able to remember to return to the postponed subproblems. If the designer fails to do so, the partial solutions could be incomplete and could not be merged or expanded into a complete solution. This problem is especially acute if the designer does not have specialized design schemas for the given problem, since its structure acts as an aid for the storage and retrieval of intermediate solutions. However, failing to return to a postponed subproblem is not always detrimental. A designer may uncover a new solution or adopt a new strategy that makes useless returning to a postponed subproblem. Nevertheless, this is different than failing to return to a subproblem because one forgets to look at one's mental notes or external notes or because these mental notes or external notes are insufficient.

In the following excerpt, P8 explicitly mentions an aspect of the problem he plans to return to. He forgets to return to it as he concentrates on the decomposition of the problem into requests from floors and requests from lifts and on merging the partial solutions. Note here that postponing that subproblem is appropriate because it is at a lower level of detail than the other subproblems he is handling. However, the postponed subproblem is crucial for the complete solution.

P8- *"It seems like in my interrupt system I don't just want a way of sequentially handling requests. I'm really going to need to be able to scan all outstanding requests because of the service constraints. I think I'll do that next. Capture these constraints in some way."*

**Difficulty in expanding or merging partial solutions into a complete solution.** The designers in our study had difficulties expanding their solution, from 1 lift to N lifts, or in merging their partial solutions, handling requests from lifts and handling requests from floors. Kant and Newell (3) also observed that the merging of the individual solutions was very difficult in the convex hull problem. Once acceptable solutions have been reached for some or all of the subproblems, these solutions must be merged together or expanded to compose a solution for the complete problem. This expansion relies on a history of the design process, and general and specialized computer science knowledge. The expansion of partial solutions actually seems difficult for most designers (this was also observed in Kant and Newell (3)). We hypothesize that the expansion of or merging of partial solutions is more difficult than the decomposition of the problem into smaller problems for three reasons:

1. Evaluative simulations for the merged or expanded solutions are more complex, more taxing cognitively than evaluative simulations for the partial solutions to the subproblems. This is because merged or expanded solutions are more complex, they have more solutions components and more interactions between these components than partial or reduced solutions.
2. Certain problem decompositions based on obvious or surface features of the problem environment (e.g., number of lifts - solve for 1 lift and expand to N lifts) may suggest solutions that do not reflect the structure of the original problem and for which no simple solution expansion exists. For example, in the case of the 1-lift problem there are no notions of coordination, communication between lifts, and race condition, which are crucial in the N-lift problem.
3. One view of the progression from design to solution/implementation is of a process of *dispersion*. When a designer breaks the problem into subproblems and refines the solution for each subproblem into more implementation-oriented representations, the implementation-oriented concepts blur the structure the original problem to solve. The solutions to the subproblems are then more difficult to merge because their original correspondence to each other has been altered. This should be more acute for complex problems where the solution of individual subproblems extends over a long period of time.

There is possibly a fourth reason. If a designer does not follow balanced development, the partial solutions will not be at the same level of detail and will be difficult to merge. This could happen for example to P8 who performed serendipitous design. However, it is difficult to precisely define when partial solutions are at the same level of detail and can, as a consequence, be mentally simulated. Therefore, it is difficult from the protocol to identify whether two partial solutions could not be merged because they were not at the same level of detail.

In the following excerpts from P8, each separate segment comes from different times in chronological order in the session and they capture the difficulty of merging the solution for one lift into a solution for N lifts.

*"...I'm basically considering what's happening for one lift. I'm imagining that everything happens independently. That's not a good assumption to make. I guess I'm considering this a global list."* He then attempts to solve the scheduling of requests and to perform evaluative simulations for N lifts.

*"...Well the fact that I have N lifts makes it kind of complicated. So I'll just try to consider the case of one lift, try to simplify it... I'm going to back up now and try to handle this with one lift."*

## RELATIONS TO OTHER STUDIES AND OTHER OF OUR FINDINGS

We will now relate some of our breakdowns to other of our findings or to other studies. Adelson and Soloway (12) studied expert designers (about eight years of professional experience) in three contexts: 1) designing an unfamiliar artifact in a familiar domain; 2) designing an unfamiliar artifact in an unfamiliar domain; and 3) designing a familiar artifact in a familiar domain. They also studied two novice designers, with less than two years of experience as designers. If one were trying to compare our designers to those of Adelson and Soloway's study, we could tentatively define: 1) P6, an expert designing an unfamiliar artifact in a familiar technical domain; 2) P8, an expert designing an unfamiliar artifact in an unfamiliar technical domain; and 3) P3, an expert/novice designing an unfamiliar artifact in an unfamiliar technical domain.

The lack of specialized design schemas seems the primary breakdown to alleviate. The reasons are both empirical and logical. P6's design solution was considered superior to P8's and P3's. P6 also exhibited a more balanced systematic design process than P8 and P3. P6 appeared to have more relevant specialized design schemas than P8 and P3. These design schemas are provided by his expertize in a relevant technical domain, i.e., communication systems. Design schemas are assumed to represent a plan to decompose a problem into subproblems. As a consequence we believe that design schemas underly balanced development. During balanced development, which was described by Adelson and Soloway (2), each solution component is developed at a similar level of detail to permit mental simulations.

Design schemas are also believed to provide a cognitive structure to help store partial solutions and their evaluations and to help remember which subproblem to focus on next. As a consequence, they are believed to support mental note-taking. Mental note-taking was observed by Adelson and Soloway (2) and described as a mechanism supporting balanced development. As a consequence, a lack of specialized design schemas will worsen another breakdown, the difficulty in keeping track and returning to postponed subproblems. This breakdown was rarely observed in P6 but was very frequent in P3.

The lack of relevant specialized design schemas appears to be associated with another breakdown, difficulty in expanding a reduced solution or merging of partial solutions. A design schema provides a plan for a well-motivated decomposition of the problem into subproblems. The partial solutions from such decompositions can be easily merged together to form a complete solution. When a problem is not decomposed on the

basis of a design schema, the partial solutions may be difficult to expand or merge together. P6 had very little difficulty merging his partial solutions, while this was particularly difficult for P8.

Finally, we believe that the use of specialized design schemas frees the designers from reliance on weak problem-solving methods such as generate and test and means-ends analysis. These weak problem-solving methods, especially generate-and-test, can be very taxing cognitively. In fact, P3 had the least relevant specialized design schemas and adopted a "generate-and-test-and-debug" strategy. Not surprisingly, he experienced many difficulties with mental simulations with many steps or many test cases.

The next important breakdown is the lack of meta-schema of the design process which leads to poor allocation of resources and time to the various activities during the design process. We believe that relevant specialized design schemas obviate the need for sophisticated meta-schemas about the design process. Specialized design schemas provide for systematic decomposition of the problem and for solutions to each subproblem. The need to carefully manage resources is less critical in this context. However, when specialized design schemas are lacking, the need to carefully balance exploration of the problem environment, consideration of alternative solutions, and evaluations of selected tentative solutions is critical for the quality of the solution reached. While P8 seemed have less specialized design schemas relevant to the problem than P6, he appeared to have a sophisticated design meta-schema, which was lacking in P3. P8's solution was considered better than P3's. P8's design meta-schema guided him to explore the problem environment before focusing a testing and debugging solutions. We speculate that the exploration of the problem environment induced *serendipitous problem-solving activities*. By serendipitous design, we mean a design process controlled by recognition of **partial solutions, at different levels of detail or abstraction,** without having previously decomposed the problem into subproblems. We hypothesize that serendipity in design arises from a form of data-driven processing as opposed to goal-directed processing, such as described by Anderson (13). This data-driven processing can be triggered by aspects of the problem environment at different levels of detail or abstraction. The concept of serendipity in design is similar to the idea of *opportunistic* problem-solving (Hayes-Roth and Hayes-Roth (14)), though different in some crucial aspects. The main difference is that in serendipitous design the partial solutions are not synergetic, they do not necessarily interact and they do not fulfill more goals than originally anticipated. An important observation to make is that serendipitous design does not follow balanced development. Moreover, P8 did not extensively exhibit mental note-taking. However, we believe that when specialized design schemas are lacking, the presence of a design meta-schema supporting important exploration of the problem environment is advantageous. We speculate that exploration of the problem environment induces serendipitous design, as opposed to balanced development. However, in the absence of specialized design schemas we believe serendipitous design is advantageous. In fact, serendipitous design might be hallmark of an important type of design, designing new innovative software systems. For such systems are not simply modifications of previously well understood systems, they introduce genuinely new ideas. As a consequence, their design cannot rely on computer science knowledge embodied in specialized design schemas. Moreover, because of real-life constraints designers may often face the situation of designing systems outside their areas of expertize, that is, for which they lack specialized design schemas. Finally, a certain amount of exploration of the problem environment is always desirable as it permits uncovering critical missing information in the requirements. In all these cases, we need to develop tools and methodologies which permit designers to benefit as much as possible from serendipitous problem-solving, as opposed to discourage it because it does not follow the prescriptive practices from software engineering.

While this is a preliminary study, it is very encouraging to see that similar design processes and breakdowns have been observed in other very different fields. After the completion of this study, we became aware of the work by Ullman, Stauffer, and Dietterich (5) on the mechanical design process. They collected verbal protocols from four professional engineers working on problems related to their areas of expertize. Our designer P8 performed a great deal of serendipitous problem-solving, probably underlied by his extensive exploration of the problem environment. Ullman, Stauffer, and Dietterich labelled these problem solving activities opportunistic. We believe that the name opportunistic is misleading, as synergy did not appear in the partial solutions reached by the designers. The partial solutions did not satisfy unanticipatedly more goals than they were originally meant to. Interestingly, Adelson and Soloway (12) described some design behaviors which may be related to serendipitous design. The particularly relevant observation is that the **experienced** designer designing a **familiar object** in a **familiar technical domain** departed from balanced development and systematic expansion of his solution when dealing with the aspect of the problem that was unfamiliar to him, the functionality of a particular chip.

Ullman, Stauffer, and Dietterich also observed that not all designers followed balanced development (even though all their designers were experienced). They observed the use of diagrams to prevent breakdowns from difficulty of mental simulations, even though diagrams were only partly successful. They also observed that designers forgets to return to postponed subproblems. They also noticed that designers have a tendency to elaborate mainly one main design idea throughout the design session with few considerations of major changes to the basic design.

So, it appears that experienced designers are aware of the power of exploration of the problem environment in uncovering new important information (i.e., as part of their meta-schema about the design process). They use exploration of the problem environment when dealing with unfamiliar information or when progress toward a solution is insufficient using balanced development. This exploration of the problem environment induces problem-solving that does not follow balanced development but is more appropriately described as serendipitous.

Similarly, Flower and Hayes and their colleagues have observed rapid shifts between levels of abstraction and detail in the planning and writting of documents and the use of design schemas and of meta-schemas (15). Finally, Schoenfeld (16) has observed similar behaviors in mathematical problem solving.

## CONCLUSIONS

While the study reported in this chapter is an exploratory study, it provides a wealth of observations that enrich our understanding of the software design process by individuals. Our findings were related to previous studies of the design process by individuals and revealed new behaviors and some important new questions about the design process.

Our observations of designers working on the N-lift problem show that designers use a wide variety of design strategies, both between and within designers, in addition to the top-down refinement approach described in software engineering. We also found that our designers were able to work at different levels of abstraction and detail and not just follow a balanced development strategy. We also observed serendipitous problem-solving, not reported in previous studies of software design by individuals. We also observed a great emphasis on understanding and elaborating the requirements through mental

simulations. We have observed a wide variety of breakdowns, also not reported or emphasized in previous studies: 1) lack of specialized design schemas; 2) lack of a meta-schema about the design process leading to poor allocation of resources to the various design activities; 3) poor prioritization of issues leading to poor selection of alternative solutions; 4) difficulty in considering all the stated or inferred constraints in defining a solution; 5) difficulty in keeping track and returning to subproblems whose solution has been postponed; 6) difficulty in performing mental simulations with many steps or test cases; and 7) difficulty in expanding or merging solutions from individual subproblems to form a complete solution.

There was also overlap between our findings and the findings of other studies in varied fields (e.g., mechanical engineering and mathematical problem-solving). This overlap suggests that we are tapping general problem-solving strategies for design problems. This also suggests that the tools we are designing to alleviate the breakdowns have wider applicability than software design. This overlap also increases confidence about the validity of the findings and their generalizability.

For each breakdown, we have recommended software tools or methodologies to alleviate them (Guindon, Krasner, and Curtis (17)). In further studies, we will test the effectiveness of these tools and methodologies. These further studies will be indirect tests of the hypotheses raised in this study. In addition, they will provide empirical results on the influence of software tools on the design process. We will also explore in greater depth the nature of the application-specific design schemas, their role in design performance, their role in the different expertize exhibited by our designers, their relation to the breakdowns we have observed, and how software tools could be designed to alleviate lack of design schemas.

## REFERENCES


1. Jeffries, R., Turner, A.A., Polson, P., & Atwood, M.E. (1981). The Processes Involved in Designing Software. In J.R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, N.J.: Erlbaum, 225-283.
2. Adelson, B. & Soloway, E. (1985). *A Cognitive Model of Software Design*. Technical Report 342, Department of Computer Science, Yale University.
3. Kant, E. & Newell, A. (1984) Problem Solving Techniques for the Design of Algorithms. *Information Processing and Management*, 28(1), 97-118.
4. Carroll, J.M., Thomas, J.C., & Malhotra, A. (1979). Clinical-Experimental Analysis of Design Problem Solving. *Design Studies*, 1(2), 84-92.
5. Ullman, D.G., Stauffer, L.A., Dietterich, T.G. (1987) Preliminary Results of an Experimental Study of the Mechanical Design Process. Proceedings of the Workshop on the Study of the Design Process. Oakland, California.
6. Jackson, M. (1983). *System Development*. Englewood Cliffs, N.J.: Prenctice Hall.
7. Laird, J., Rosenbloom, P., & Newell, A. (1986). *Universal Subgoaling and Chunking*. Kluwer Academic Publishers.
8. Rich, C., & Waters, R.C. (1986) Toward a Requirements Apprenctice: On the Boundary Between Informal and Formal Specifications. M.I.T. A.I. Memo No. 907.
9. Lubars, M.T., & Harandi, M.T. (1987) *Knowledge-Based Software Design Using Design Schemas.*" Proceedings of the Ninth International Conference on Software Engineering, Monterey, California., pp 253-262.
10. Tong, C. (1984). Knowledge-Based Circuit Design. Ph.D. Dissertation, Department of Computer Science. Stanford University.

11. Mostow, J. (1985). Toward Better Models of the Design Process. *AI Magazine*, 44-57.
12. Adelson, B. & Soloway, E. (1985). The Role of Domain Experience in Software Design. *IEEE Transactions on Software Engineering, Vol. 11, No. 11.*
13. Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.
14. Hayes-Roth, B. & Hayes-Roth, F. (1979). A Cognitive Model of Planning. *Cognitive Science*, 3(4), 275-310.
15. Flower, L., Hayes, J.R., Carey, L., Schriver, K., Stratman, J. (1986). Detection, Diagnosis, and the Strategies of Revision. College Composition and Communication, Vol. 37, No. 1.
16. Schoenfeld, A.H. (1987) *Mathematical Problem Solving.* Academic Press.
17. Guindon, R., Krasner, H., Curtis, B. (1987). A Model of Cognitive Processes in Software Design: An Analysis of Breakdowns in Early Design Activities by Individuals. MCC Technical Report in preparation.

## ACKNOWLEDGEMENTS

We wish to thank Glenn Bruns, Jeff Conklin, Michael Evangelist, and Colin Potts for very insightful comments and criticisms on an earlier version of this paper.

## APPENDIX

### PROBLEM STATEMENT

An N-lift (N-elevator) system is to be installed in a building with M floors. The lifts and the control mechanism are supplied by a manufacturer. The internal mechanisms of these are assumed (given) in this problem.

DESIGN THE LOGIC TO MOVE LIFTS BETWEEN FLOORS IN THE BUILDING ACCORDING TO THE FOLLOWING RULES:

1. Each lift has a set of buttons, 1 button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e. stopped at) by the lift.
2. Each floor has 2 buttons (except ground and top), one to request an up-lift and one to request a down-lift. These buttons illuminate when pressed. The buttons are cancelled when a lift visits the floor and is either traveling in the desired direction, or visiting the floor with no requests outstanding.
   In the latter case, if both floor request buttons are illuminated, only 1 should be cancelled. The algorithm used to decide which to service first should minimize the waiting time for both requests.
3. When a lift has no requests to service, it should remain at its final destination with its doors closed and await further requests (or model a "holding" floor).
4. All requests for lifts from floors must be serviced eventually, with all floors given equal priority (can this be proved or demonstrated?).
5. All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel (can this be proved or demonstrated?).
6. Each lift has an emergency button which, when pressed causes a warning signal to be sent to the site manager. The lift is then deemed "out of service". Each lift has a mechanism to cancel its "out of service" status.