Designers' activities examined at three levels:

organization, strategies problem-solving

Willemien Visser

This test is a pre-print of Visser, W. (1992). Designers' activities examined at three levels: organization, strategies & problem-solving. *Knowledge-Based Systems, 5*(1), 92-104.

DESIGNERS' ACTIVITIES EXAMINED AT THREE LEVELS: ORGANIZATION, STRATEGIES AND PROBLEM-SOLVING PROCESSES

W. Visser

Institut National de Recherche en Informatique et en Automatique

(National Institute for Research on Computer Science and Automation)

Projet de Psychologie Ergonomique pour l'Informatique

(Ergonomics Psychology Project)

Rocquencourt B.P.105

78105 LE CHESNAY CEDEX (FRANCE)

email: willemien.visser@telecom-paristech.fr

Table of contents

This test is a pre-print of Visser, W. (1992). Designers' activities examined at three levels: organization, strategies & problem-solving. *Knowledge-Based Systems, 5*(1), 92-104.

DESIGNERS' ACTIVITIES EXAMINED AT THREE LEVELS: ORGANIZATION, STRATEGIES AND PROBLEM-SOLVING PROCESSES

W. Visser*

Institut National de Recherche en Informatique et en Automatique

(National Institute for Research on Computer Science and Automation)

Projet de Psychologie Ergonomique pour l'Informatique

(Ergonomics Psychology Project)

Rocquencourt B.P.105

78105 LE CHESNAY CEDEX (FRANCE)

email: willemien.visser@telecom-paristech.fr

Abstract. Through examples of data from three empirical design studies, this paper presents the type of analysis which cognitive psychology makes of the mental activities involved in design. These activities are analyzed at three levels: the way in which designers organize their activity, the main strategies they adopt, and the problem-solving processes they use.

Different types of design tasks are presented: functional specification, software design and composite-structure design tasks.

The relevance of the results for A.I. is discussed from an assistance viewpoint.

Keywords. Design, Organization, Planning, Strategies, Problem-solving processes, Cognitive psychology, Assistance systems, A.I.

* Thanks to A. Bisseret, B. Trousse, P. Falzon, F. Darses and R. James for their comments on a previous version of this paper.

This test is a pre-print of Visser, W. (1992). Designers' activities examined at three levels: organization, strategies & problem-solving. *Knowledge-Based Systems, 5*(1), 92-104.

INTRODUCTION

This paper addresses one of the aspects of design -or rather of "real" problem solving in general- which Donald Schön considers as particularly important, but as distorted by the approach which most researchers take to design: the constructive character of the design problem solving activity. The three studies presented in this paper show indeed that a problem solver is not "given" problems, but that he "constructs" them. Our results confirm that design problems are not "given" in the problem specifications, neither in the sense of fixed right from the start, nor in the sense of already defined by another person, e.g., the client. The analyses of design problem solving made by Schön insist particularly on the role of the situation, whereas our studies focus on the different types of design knowledge (both in the domain of action execution and in that of action management) which are used by the designer when he takes into account the different data which he has at his disposal and which he receives from other sources.

The perspective which we take towards A.I. is also the same as Schön's, i.e., a design assistance, not a design automation viewpoint. We consider that, in order for design assistance systems to be likely to really support designers, they are to be based on data on the activity to be assisted. Cognitive psychology research such as presented in this paper may provide such data!

Design studied from the cognitive psychology viewpoint

Cognitive psychology research on design mainly involves the study of

- knowledge involved in designing and processing (using) this knowledge;
- the organization of the actual activity and the strategies implemented by designers.

These aspects have been studied in various design domains, even if most empirical studies have been conducted in the domain of software design (see Hoc, Green, Samurçay & Gilmore, 1990). Other domains in which design has been examined through empirical

studies are: architectural design: Eastman (1970); errand planning: Hayes-Roth & Hayes-Roth (1979); text composition: Hayes & Flower (1980); computational geometry algorithm design: Kant (1985); mechanical design: Whitefield (1986) and Ullman, Staufer & Dietterich (1987); traffic-signal setting: Bisseret, Figeac-Létang & Falzon (1988); computer-network design: Darses (1990). The studies presented in this paper have been conducted in three domains: mechanical design, software design and composite-structure design.

Design studies have mainly concerned solution development by knowledge evocation. One of our studies has examined, next to this important approach, the other main solution development mode, i.e., solution elaboration.

A final characteristic, valid of all three studies presented in this paper, is the focus on dynamic -rather than static- aspects of problem solving, i.e., planning of the design activity, strategies implemented and problem-solving processes used in order to execute the design task.

Organization of the paper

A short section presents some important cognitive psychology concepts with which the reader should be familiar in order to have an appropriate understanding of this paper (Section 2).

In the next sections, the cognitive aspects of design will then be analyzed at three levels: the global organizational, the strategic, and the individual problem-solving process level. These analyses will be discussed through the presentation of three studies conducted on three different types of design tasks:

- a study on the design of functional specifications will be used to present the global organization of a design activity (Section 3);
- a software design task will illustrate expert design strategies (Section 4);
- a composite-structure design task has been analyzed with respect to different types of problem-solving processes in design (Section 5).

The final section will show how the results of these studies are relevant for A.I.

EMPIRICAL DESIGN STUDIES

Two distinctions which are important to be taken into consideration when one conducts empirical design studies will be shortly discussed here. These are the distinctions "Task vs. Activity" and "Verbalization vs. Introspection." They are discussed only with respect to some aspects which we consider particularly relevant for a good understanding of the studies presented in this paper.

Task vs. Activity

The concept "task" refers to, either what subjects are supposed to do (their "prescribed" task, as it has been specified by their manager, by instructions or manuals), or the task they set to themselves (their "actual" task). It is only seldom that these two coincide: one of the characteristics which are often observed on a subject's "actual" task is the introduction into their task by the subject of constraints which had not been prescribed and the allocation of different degrees of importance to those which had been prescribed.

"(Cognitive) activity" refers to the way subjects actually realize their task on a cognitive level: the knowledge and other information sources they use, the way they make use of them (and of other tools) and other reasoning processes, and their intermediary and final productions.

A person can provide -e.g., in an interview- data on their task; however, many data on relevant cognitive aspects of their activity cannot be provided explicitly by persons themselves. These data are generally to be inferred, from indirect activity traces, such as verbalizations, observational data or (other) experimental data, which are to be collected by a researcher in the position of an external observer (cf. below).

The results of the first study presented below provide an example of an important difference with respect to different aspects of the design activity between the data which may be collected in interviews and those which may be collected by real-time observations.

Verbalization vs. Introspection

"Verbalization" is a data-collection method often used in cognitive psychology studies of problem solving (see Ericsson & Simon, 1980, 1984). Discussion is restricted here to "simultaneous verbalization," the method which has been used in all three studies presented in this paper.

The researcher asks the subjects to "verbalize their thoughts" or to "think aloud," i.e., to report all their mental activity, the information they take into consideration, the choices they are confronted with, the criteria used to take a decision, their reasoning, their hesitations, their questioning past decisions, etc..

There is an essential difference with "introspection," a method which was frequently used around the century -and which is still often the main data-collection method used by people developing methods or assistance tools (e.g., computer scientists, design methodologists).

Introspection consists in commenting on one's own mental activity, whereas what is asked during "simultaneous verbalization" is to verbalize the information attended to, i.e., the information which has been generated by the task-directed processes. If researchers are interested in a mental activity, e.g., the problem-solving activity underlying a design activity, they do not want to collect comments on, opinions about, or rationalizations of the mental activity. The data they want to obtain are the direct traces of the information actually used in the mental activity, because they are indirect traces of the internal stages of the cognitive process underlying the activity. The next step involves analyzing these data, according to strict rules and methods. The researchers do not want the "subjects to speculate and theorize about their processes … [They want to leave] the theory-building part of the enterprise to the experimenter. There is no reason to suppose that the subjects themselves will or can be aware of the limitations of the data they are providing [when probed to proceed to introspection]." (Ericsson & Simon, 1980, p. 221)

The use of verbalization is submitted to several conditions. The two most important ones are expressed by the following summary given by Ericsson & Simon of the effects which the instruction to verbalize may have on the cognitive process one is interested in.

"Producing verbal reports of information directly available in propositional [i.e., language or verbal] form does not change the course and structure of the cognitive processes. However, instructions that require subjects to recode information in order to report it may affect these processes." (p. 235)

"Only information in focal attention can be verbalized." (ibid.) "Automation …. [makes] the intermediate products [of task-directed information processing] unavailable to STM [short-term memory], hence unavailable for verbal reports." (p. 225)

ORGANIZATION OF THE DESIGN ACTIVITY[1]

Focus on plan deviation. Early empirical design studies, especially in the domain of software design (see Visser & Hoc, 1990), generally characterize the human design activity as being hierarchically organized, in other words, following a pre-established plan. They assert that designers' global design control strategy consists in decomposing their problem according to a combined top-down, breadth-first strategy.

Our observations in preliminary studies led us to become slightly suspicious of these conclusions: we observed top-down and breadth-first decomposition strategies to be implemented only locally, in other words, their combination did not seem to be the control strategy of the design activity at the global-organizational level. In an attempt to clarify this point, we examined the global organization of the design activity, focusing on possible deviations from a pre-established plan.

[1] This section presents in a somewhat revised way results presented in Visser (1988, 1990b).

Method

Task: Specification. A mechanical engineer had to define the functional specifications for the control part of an automatic machine tool.

Subject: A mechanical-design engineer. The observed engineer had more than ten years of professional experience in the machine-tool factory where he was working.

Data collection: Observations & Simultaneous verbalization. During a period of three weeks, full time observations were conducted on the engineer involved in his task.

The engineer's normal daily activities were observed without any intervention, other than to ask him to verbalize his thoughts during his problem-solving activity as much as possible (cf. the distinction "Verbalization vs. Introspection" discussed in the previous section).

Notes were taken on the designer's actions; all documents which he produced during his work were collected.

Results. Preliminary remarks

The engineer's global plan. The engineer, when asked to describe his activity, presented it as following a hierarchically structured plan (see Visser, 1988, 1990b). In this text, this plan is referred to as the engineer's "global plan," as opposed to local plans which the engineer was observed to formulate and/or to implement.

Notwithstanding this global plan which the engineer claims to follow, the actual activity which we observed is in fact opportunistically organized. Only as long as they are cognitively cost-effective, the control selects for execution actions proposed by the global plan ("planned" actions). As soon as other actions are more interesting from a cognitive-cost viewpoint, a deviation from the global plan occurs in favour of these actions ("deviation" actions) (see Visser, 1991b).

N.B. This result is an interesting example of the difference which may exist between the actual activity of a designer (on which data may be collected by observational methods) and

the description they may give of it (during interviews). The description which corresponds to the designer's representation of their activity is observed to have a rather different structure than the designer's actual activity!

Deviation action - Planned action. Two terms will be used as shorthand:

- "deviation action" for "action proposed by a knowledge source as an alternative to the planned action and selected by the control (leading to plan deviation);"
- "planned action" for "action proposed by the global plan and selected by the control."

Specification action - Design component. Specification proceeds by " specification actions" which consist in defining "design components" on several "descriptors."

The design components to be defined are machine-tool cycles or elements of these cycles, i.e., machine-tool functions or machine-tool operations.

The "descriptors" are attributes on which the design components are defined. The most important ones are: "identifier," "duration," "starting conditions," "ending conditions," and "physical device."

A blackboard model of the specification activity. Various authors have shown blackboard models to be particularly appropriate for modeling opportunistically organized activities (see Bisseret, Figeac-Létang, & Falzon, 1988; Hayes-Roth & Hayes-Roth, 1979; Whitefield, 1986). We chose such a model for the specification activity.

As the organization of the specification activity is the focus of this section, the adopted control structure will be briefly presented.

Control structure. Specification actions are articulated according to the following iterative sequence:

(a) Action proposal: one or several knowledge sources make action proposals because they are able to contribute to the resolution of the problem as it is defined by the state of the blackboard;

(b) Action selection: one of the proposed actions is selected by the control, depending on • the state of the blackboard;

• the action-proposing knowledge sources, i.e., the global plan and other action-proposing processes; and

• the control knowledge (especially particular selection criteria);

(c) Action execution: the selected action is executed, modifying the state of the blackboard;

(d) back to (a).

The steps which are relevant from an organizational viewpoint, that is, steps (a) and (b), guide the following presentation of results.

Results 1. Action proposal. Processes leading to deviation-action proposals

Two types of knowledge sources propose specification actions: the global plan and other action-proposing processes. As mentioned above, this section focuses on the "deviation-action proposing processes." Five of them have been identified through an analysis of the specification actions which did not match the engineer's global plan.

Taking advantage of available information

Acting according to a plan is a concept-driven activity. If the engineer follows his global plan, each step of the plan will propose and impose a goal which will make him look for the information needed to achieve this goal.

Deviation, however, is generally due to data-driven processing. The most general form of deviation stems from the engineer trying to achieve the goal which available information allows him to achieve. This goal generally differs from the one imposed by the global plan.

Information may be available because the engineer is "presented" with it, or because it is being used for the current specification action.

Some examples of "information presentation" are the following:

• the client communicates to the engineer new or modified requirements;

• a colleague presents the engineer with information or comments on the solution state;

• a consulted document presents the engineer with "salient" information (information generally is "salient" because it is new or modified).

The next three processes presented below are different forms of the second case, that is, different processes for taking advantage of information used for the current specification action.

Processing information from various viewpoints leading to its use for an action other than the current one

Information used to define a component may be processed from a viewpoint that makes it useful in other specification actions or even a task other than functional specification.

This process was most frequently observed as leading to deviations from one specification action to another.

Example. Information used for defining the starting conditions of an operation was often interpreted as also being useful for defining the ending conditions of one or more previous operations.

The process sometimes led the engineer to deviate from the functional specification to another specification task. Information used for functional specification was considered, e.g., from its mechanical, i.e., physical viewpoint.

Example. During his definition of a turntable cycle operation, the engineer consults a mechanical specifications document that includes information on the electrical detectors of the turntable. In processing this information, the engineer comes to question the mechanical safety of the turntable: he is not sure that the turntable will not damage other machine-tool stations during its rotational movement. So he interrupts his turntable cycle definition (a functional specifications subtask) in order to verify the mechanical specifications. Thus, he interrupts his functional-specification activity to proceed to a different task, that is, mechanical design.

Activation of a component representation other than the current one leading to a local plan for defining another component

In order to define a component Cx, its mental representation RCx has to be processed, leading to its activation in memory. This activation, in turn, may lead to the activation of the representation RCy of another component Cy, by the mechanism of "spreading activation" (see Anderson, 1983) (see Figure 1).

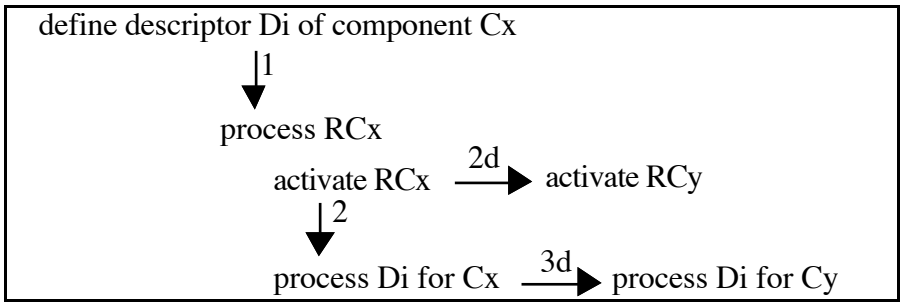

Figure 1.Two processes leading to deviation-action proposals

Key: vertical arrows: planned actions
horizontal arrows: deviation actions

The main cause of RCx activating RCy -rather than, for example, RCz- is a relationship existing between RCx and RCy (in the designer's knowledge). Four types of these "activation-guiding" relationships have been identified: analogy, prerequisites, interaction, and opposites.

Analogy. The machine tool has two workstations: shaping and finishing. When the engineer is dealing with a component for one of these stations, he often returns to, or anticipates, the definition of the corresponding component in the other station, because he considers their functioning to be analogous. This is reflected in the relationship between their corresponding mental representations (see Visser, 1991a, for a detailed analysis of this use of analogy).

Prerequisites. All work operations (doing the shaping and the finishing on the machine-tool workstations) have prerequisites, that is, operations which satisfy the conditions required for these work operations to take place (e.g., positioning a tool, or setting a stop to establish the position where a work operation must end).

The engineer was observed to "forget" certain operations, that is, he forgot to introduce them during the first definition pass and introduced them only afterwards. With one exception, these "forgotten" operations were prerequisites. The engineer generally discovered their omission when he was specifying the work operation of which it was a prerequisite. So processing a work operation led to the activation of the representation of its prerequisite.

Interaction. This relationship may be based on physical or functional relations between components.

Between machine-tool units, existing relations are generally physical. The turntable, for example, interacts with each of the other units, since these must be retracted before it can turn. The engineer partially describes the turntable cycle before the cycles for the other units. When dealing with certain of them, he goes back to the turntable cycle specification in order to introduce information in it.

Functional relations are generally between operations. For example, the starting conditions for an operation always comprise one or more of the ending conditions of the preceding operation(s). This two-directional relationship leads to two different deviation patterns. Firstly, the processing of the starting conditions for an operation has been observed to lead to the discovery and fixing of omissions or errors in the ending conditions of the preceding operation. Secondly, the processing of the ending conditions for an operation was observed to lead the engineer to jump ahead to the following operation, in order to define its starting conditions in anticipation.

Opposites. When dealing with an operation, the engineer sometimes discovers that he has omitted its opposite operation.

Definition of a component on a descriptor leading to a local plan for defining other components on the descriptor

The deviations presented above were based on exploiting semantic relationships between design component representations. Deviation may also be caused by the exploitation of the particular type of definition one is engaged in. In this case the design component the deviation leads to has no relationship with the currently defined design component other than belonging to the same cycle. Defining a component on one of its descriptors leads the engineer to formulate a local plan for defining one or more other components on the same descriptor (see above, Figure 1).

When this process leads to deviation, it frequently leads to several consecutive deviation actions. This may occur in several ways:

• having defined a design component Cx on a descriptor Di leads the engineer to define another component Cy also on Di; next, the engineer resumes his global plan and defines Cx on Dj, which may lead him to deviate again in order to define Cy also on Dj, etc.;

• having defined a design component Cx on several of its descriptors, Di to Dn, leads the engineer to define another component Cy on these same descriptors, Di to Dn;

• rather than deviate to only one other design component, several other components may be defined (on one or several descriptors) before the global plan is resumed.

Difficulties with the current specification action leading to "drifting"

"Drifting" refers to involuntary attention switching to a processing other than the current one. It was observed to occur especially when the engineer was involved in a "difficult" action (see below). During such an action, all attention is supposed to be focused on the current problem. Drifting, under these conditions, may be due to the engineer

• "looking in various directions" for possible solution elements, then

• coming upon information which more or less "obviously" applies to another design component definition, and then

• taking advantage of it.

The engineer may look for possible solution elements at various places, in various information sources. If he "looks" in memory, in other words, if he constructs a representation of the problem which he is trying to solve, this activates the mental representational entities of which the problem representation is built up. This activation may activate other entities via the relationships between them.

A hypothesis inspired by our observations is that drifting-caused deviation occurs especially during information retrieval for problem solving that is not guided by strict information-searching rules.

Results 2. Action-selection: a combination of control criteria

To decide which action, among the proposals, is going to be selected for execution, the control evaluates each proposed action according to selection criteria and compares the proposals on these criteria.

The global plan always proposes one action, but often there are several competing actions when, next to the planned action, one or various deviation proposals have been made.

Two selection criteria have been identified: cognitive cost and importance of the action.

First action selection criterion: Cognitive cost

For each proposed action, the cognitive cost is determined. For a planned action, this is its cost if-executed-now compared to its cost if-executed-later. For a deviation action, its cost if-executed-now includes, in addition, the cost of plan resumption.

In evaluating the cognitive cost of an action, the control considers several, not necessarily independent, factors, three of which are presented below.

The availability of a schema for executing the action. A schema provides, for each of its variables, a certain number of values, generally including one default value. Executing an action for which such a memory representation is available may cost relatively little if most or all variables relevant for execution have default values.

The engineer's global plan may be considered to be a schema of this nature. This is why its use is so profitable from the cognitive-cost viewpoint. But, as shown above in the presentation of processes leading to deviation actions, the designer also has other schemata linking together design or application-knowledge representational entities. Use of these schemata, i.e., activation via their relationships, may lead to deviations from the global plan when the resulting actions are more profitable from a cognitive-cost viewpoint.

The availability of information. On the one hand, an action may be interesting from the cognitive-cost viewpoint if the information required for executing it is available without much effort, because the cognitive cost of information access affects the cognitive cost of the action. On the other hand, an action may become interesting because available information allows it to be executed.

The difficulty of the action. Actions are more or less "difficult" depending on the number - and perhaps the nature- of their component operations. To retrieve a value is easier than to calculate a value, because a calculation still requires its constituent values to be obtained. Calculation is an operation that is itself more or less difficult depending on the availability of the elements used in the calculation.

Example. The duration of an operation can often be found as such in a document. If not, it is to be calculated from its constituents, motor speed and advance distance/rotation. Sometimes, even these are not given in the engineer's current information sources, and still have to be looked up in technical documentation.

Second action selection criterion: Importance

Actions differ with respect to their importance. The two contributing factors which have been identified will be presented through an example.

The importance of the type of action. Example. Fixing the omission of an operation which has been forgotten is an important action; verifying an operation identifier is not.

The importance of the object concerned by the action. Example. If verifying is not an important action when it concerns identifiers, it is when it concerns durations. The engineer frequently deviates from his global plan in order to verify the duration of an operation, but never to verify its identifier.

Combination of the criteria

Among the two criteria presented above, the first one, "cognitive cost," is considered to be the most important and most general one. Our hypothesis with respect to the way in which the two criteria are combined is the following:

(a) If the next planned action is the only proposed action and if it does not cost too much, it is selected for execution. If it costs too much, it is postponed and the next proposed actions are taken into consideration.

(b) If one deviation action is proposed as an alternative to the next planned action, their cognitive costs are compared: the action which is lowest in cost is selected for execution.

(c) If several deviation actions are proposed as alternatives to the next planned action, their cognitive costs are compared: if there is one action that is lowest in cost, it is selected for execution; if several actions have the same cost, the most important one is selected.

Comparative evaluation of the costs of an action: if-executed-now and if-executed-later

In order to select the action to be executed, control evaluates and compares the proposed actions with respect to their cognitive cost (and their importance). But the cost of a proposed action, i.e., the cost of the action if-executed-now, can also also compared to the cost of the action itself if-executed-later. This may lead to different types of deviation:

(a) Anticipation. A proposed deviation action D is chosen in anticipation because its relative cost is especially interesting compared to D when it will be the planned action. Of course, in order for this deviation action to be selected, it should be not (much) more expensive than the planned action.

(b) Postponement. The planned action P may be the only action that is proposed and still be postponed, because it costs too much, not compared to a currently proposed deviation

action, but to P itself if executed later. In this case, a local plan is generally formed for re-proposing the postponed action, either as soon as the conditions leading to its postponement no longer prevail, or at a given moment later in the design process.

A reason for a planned action costing too much if executed now, is the required information being unavailable (now). This may be the case, for example, because

• the client has not yet made a decision that is necessary for the definition of the component in question (and the engineer judges that he cannot make this decision himself);

• the information source with the relevant information (usually a colleague) is absent.

DESIGN STRATEGIES[2]

This section presents design strategies observed on a professional software engineer, focusing on those strategies which are specific to designing software in a work context.

Our study was guided by the hypothesis that, from a strategical viewpoint,

(a) design cannot be characterized appropriately as following one pre-established global strategy (such as following a plan in a top-down, breadth-first order);

(b) the activity is governed by various locally implemented strategies.

Another hypothesis underlying the study was that these strategies differ, at least partially, from those observed in most classical empirical software design studies. Those studies were generally conducted on novice, student programmers working on artificial, limited problems, whereas the designer in our study was an experienced, professional programmer who was observed in his daily work environment.

Method

Task: Software design. A programmer had to design and code the program for the control part of the machine tool described above, using the specifications defined by the mechanical engineer.

[2] This section presents in a completely revised, less detailed way, results presented in Visser (1987). For information about the programmable controller and its programming language, this paper may be consulted.

Subject: A software engineer. The programmer was a software engineer who had some four years experience in the design of software for automatic machine tools.

Data collection: Observations & Simultaneous verbalization. Over a period of four weeks, full time observations were conducted on the programmer involved in his task.

The procedure was the same as presented above for the mechanical-design engineer.

Results

We start by a discussion of decomposition, the strategy which, in most classical software design studies, is considered to be the designers' global control strategy. Afterwards, the main strategies which were actually used by the programmer will be presented: re-use, considering users of the system, and simulation. These three strategies, which are at different abstraction levels, were used in combination.

Other design strategies observed on the programmer and on other software-design experts may be found in Visser & Hoc (1990).

A preliminary definition. Software-design components. The programmer's design components may be program modules, sub-modules, instructions, or instruction branches (instructions are made up of several serial and/or parallel branches).

Problem decomposition

In the literature on (software) design, "problem decomposition" is a recurrent notion. For one thing, it is advocated by existing design methodologies, which provide different bases for performing it. For another, until recently, most empirical software-design studies presented it as a very important -if not the main- control strategy in expert design activity.

By definition, a design problem is "decomposed," in that it is transformed into other problems to be solved. As used in those early design studies, however, the concept conveys particular presuppositions, such as that decomposition results from "planning" and leads to

"balanced" (top-down breadth-first) solution development (see Visser & Hoc, 1990, for a critical presentation).

With respect to the decomposition of the activity, we tried to break down the programmer's activities into consecutive "stages". Four remarks may be made concerning this breakdown.

• A sequence of actions was considered to constitute one "stage" when the actions were homogeneous with respect to their function (e.g., planning, problem understanding, solution development, solution evaluation) (see also Pennington & Grabowski, 1990).

• There were no separate "designing" and "coding" "stages" in the programmer's actual activity. Therefore, the activity has been qualified as "programming."

• Using the distinction criterion, a decomposition into a first and a second stage could be neatly identified; in the second stage, a further decomposition has been observed:

(a) Problem understanding. The first day, the programmer "studied" and "analyzed", i.e., skimmed through, the specifications he had received (some 50cm of A4 documents).

(b) "Programming planning". Afterwards, during one hour, the programmer plans his programming activity (leading to a "programming plan"). He does so along two lines:

- Decomposition of his task into program parts, according to the relative urgency with which various colleagues (especially in the workshop) need the different parts. This leads the programmer to distinguish three program parts, which he plans to handle consecutively.

- Decomposition of the program into modules corresponding to machine-tool functions. This breakdown follows the order of modules on the listing of an "example" program that the programmer had previously written for a similar machine tool.

• After this planning stage, the programmer directly started to code. This coding was -of course, one should say- often interrupted for one or the other of the different types of activities distinguished above, from planning to solution evaluation, but not necessarily iteratively in a fixed order. Thus, because the interruptions were not systematic, the activity during the rest of the four weeks could not be considered as having been further decomposed into functionally homogeneous stages.

So, "planning" in this coding "stage" was "local" and "punctual". It took place at varying problem-solving levels and concerned more or less large entities (e.g., at the design level, a function or a machining operation; at the coding level, a module or an instruction). The programmer deviated from the "programming plan" which he had formulated based on the order of the example program modules if another order was judged more economical from the point of view of cognitive cost -and this often resulted in a local plan. These local deviations or alternative programming plans leading to more global deviations were triggered by various processes of the same type as those proposed above in the section on the Organization of the design activity.

Re-use

Re-use was an important strategy in the activity of the observed programmer. Re-used components may be distinguished according to their origin: programs written in the past, or the current program. Both were among the programmer's main information sources. They were used much more frequently than the functional specifications of the machine tool, that were consulted only during the first design stage.

The programmer used two "example" programs that he had previously written for similar machine tools. One was only used for the construction of one particular target-program module. The other was used very frequently, in the design stages of program-coding planning and of coding (this program is referred to as "the" example program).

Re-used components may also be distinguished according to the level at which they are used: structuring the program or coding the instructions. In the program-coding planning stage, only functions existing in the example program were planned (others appeared later during coding): the programmer listed them by copying their titles from the example program.

A re-usable component was generally accessed by way of the semantic relationship between the mental representations of the target and source components.

Examples of exploited relationships are:

• analogy between the structures of the example and target programs;

• analogy between functions of parts of the example and target programs;

• analogy between functions of parts of the target machine tool;

• opposition between functions on the target machine tool.

Considering users of the system

This strategy had different functions. It guided solution evaluation as well as solution development.

Solution evaluation. One of the criteria used by the observed programmer in his evaluation of his design was "ease of use for future users" (system operators as well as maintenance personnel).

Solution development. Considering homogeneity an important factor of ease of use, the programmer used it as a design constraint, trying to make the program as homogeneous as possible, both for comprehension (system operators) and for maintenance reasons (maintenance personnel). This search for homogeneity was realized in different ways. Two examples are the following.

Example. Creation of uniform structures at several levels of the program.

• Instruction order in the modules. Instructions corresponding to Advance operations were made to precede the related Return-movement instructions.

• Branch order in the instructions. The branch defining the automatic mode of an operation was made to precede its manual mode definition branch.

• Bit order in instruction branches. The "most important" enabling conditions of an operation were put at the beginning of its corresponding instruction branch[3].

• Variable numbering. Variables are identified by numbers. Numbering of certain variables with counterparts elsewhere in the program was structured. For example, the variables corresponding to the different "Checks" on the Return movement of each machine station, defined in different program modules, were numbered B601 on one station, B701 on another, and B801 on a third station.

[3] Instructions in programmable controller programs are scanned continuously. Inspection of an instruction by the supervisor is stopped as soon as a bit set to 0 is encountered in a serial connection branch.

Next to this intra-program, inter-program variable numbering homogenizing was also observed. Certain variables are found in nearly all machine tool programs in the factory where the observed programmer is working, that is, in his own programs, but also in those written by his colleagues. The programmer gave these variables the same numbers as they have in other programs.

Example. Program changes. The search for homogeneity sometimes led the programmer to reconsider previously written instructions. In doing so, he modified either

- the current instruction in making it follow the structure of the previously written ones; or
- the previously written instructions by giving them the structure of the current instruction.

Simulation

The observed simulation was always mental, never concrete. Within this mental simulation, different types may be distinguished. First with respect to the problem-solving stage in which simulation took place: simulation was used for problem development, when the programmer explored and simulated the problem environment; or for solution evaluation, when the programmer ran simulations of proposed solutions.

Second with respect to the object involved in the simulation. This object could be the machine tool operation: this simulation was mainly used to understand the functional specifications (problem development stage). Simulation of program execution, generally considered to characterize novices, was one of the means used by this experienced programmer to check program modules that he had already written (solution evaluation stage).

This test is a pre-print of Visser, W. (1992). Designers' activities examined at three levels: organization, strategies & problem-solving. *Knowledge-Based Systems, 5*(1), 92-104.

DESIGN-PROBLEM DEVELOPMENT PROCESSES[4]

The two previous sections have analyzed the activity at rather high levels: individual design problem solving was not addressed. That is the topic of this section, which presents design problem solving at the level of the processes used to solve an individual problem. We analyze what happens when, according to organizational or strategic considerations, a particular problem has been selected as being the next design problem to be solved.

At this level, problem solving may be considered to proceed in three steps: problem-representation construction, solution development, and solution evaluation. This section focuses on the second step: it is going to present the main types of solution-development processes involved in design problem solving.

Method

Task: Composite-structure design. The global task was to design a new type of antenna, an "unfurling" antenna.

Subject: A technician specialized in composite-structure design. The observed designer was a technician with some 30 years of professional experience in the Research and Development division of an aerospace factory. For the past four or five years, he had been involved in designing antennas.

Data collection: Observations & Simultaneous verbalization. The designer was observed over nine weeks, at the rate of 3-4 days a week. The analysis on which the presented results are based concerns the data collected during the first five weeks.

The same procedure was used as in the previous studies.

[4] This section presents, in less detail, results presented in Visser (1990a). Examples on the antenna project would have required a detailed description of the problem of "unfurling," which would have occupied too much space. Only two simple examples can be presented.

Results. Preliminary remarks

Three aspects of problem-solving which underlie the presented analysis will be briefly discussed.

Problem or Solution: A double status. Design may be considered as specifying a problem until an implementable solution has been reached. The problem which is the starting point for a designer are the specifications provided by a client, which specify more or less precisely the artefact to be designed. The solution to be attained is another series of specifications, which the designer provides to the workshop and which specify precisely how to manufacture the artefact.

The problem-solving path, consisting of a transition from the initial problem to a final solution, comprises a great number of intermediary states, each with a double status. Until a final implementable solution has been attained, each solution developed for the problem under focus constitutes, on the one hand, a further specification of this problem and, on the other hand, a new problem to be solved. That is why, in this text, the same entity may be referred to as a "solution" or as a "problem," depending on whether it is an output or an input of a problem-solving action.

Problem/Solution knowledge: Structure and access. Experts have in memory "problem/solution schemata" organizing their knowledge about classes of problems and their solutions, with "problem attributes" and "solution attributes."

However, a great part of their knowledge on problems and their solution(s) cannot -or cannot yet- be considered to be schematic: it does not concern classes of problem situations, but particular problems and their solution(s). The corresponding representations are "problem/solution associations."

In this text, except if there is evidence for problem/solution knowledge being either schematic or specific, this knowledge will be referred to as "problem/solution(s)."

The mechanism underlying the access to this knowledge is supposed to be "spreading activation" (see Anderson, 1983).

Solution evocation and solution elaboration. Solution development starts with the construction of a mental representation of the problem to be solved. The components of the resulting representation, that is, certain mnesic entities, will be the "sources" for activating other mnesic entities. If a problem/solution is matched, the corresponding solution is said to be "evoked." If this fails, elaboration of a solution is engaged. The next two sub-sections present these two main solution-development processes.

Results 1. Solution evocation

Different types of evocable solutions are distinguished according to

- their "historical link" to the current problem ("pre-existing" solutions vs. solutions developed for the current problem); and
- their likelihood to be activated ("standard" vs. "alternative" solutions).

Evocation of a "pre-existing" solution or of a solution developed for the current problem

Solutions acquired by experience exist in memory before the current global-problem solving (analogous to re-usable programs written in the past). Two types of these "pre-existing" solutions are distinguished: "standard" and "alternative" solutions.

But the current problem solving also leads to solutions being stored in memory (analogous to re-usable components of the current program). Depending on several factors, some are explicitly "shelved," whereas others will enter long-term memory even if the designer consciously did nothing to retain them. A solution is said to be "shelved" if it is developed, not positively evaluated, but -rather than rejected- put aside "for possible later usage."

Evocation of "the" "standard" solution - Evocation of an "alternative" solution

Design problems generally have more than one solution. The designer does not necessarily know of several solutions to a problem, but if he does, one of them is activated most strongly

on evocation, and so evoked first. This is "the" "standard" solution to this problem; the other(s) is (or are) the "alternative" solution(s).

If the designer knows of several solutions to a problem, the alternative solution(s) may be evoked if the standard solution is rejected after evaluation, in other words, in later solution development cycles.

Results 2. Solution elaboration

Elaborating a solution may take several forms, according to the character of the material constituting the starting point for elaboration.

(a) Elaboration of an "alternative" solution. The starting point for elaboration may be a solution which has been previously developed, but which was rejected after evaluation. This "negative" evaluation introduces supplementary constraints on the alternative solution to be elaborated. These are added to the constraints introduced by the original problem which are supposed to have already been taken into account in the previously developed solution.

(b) Elaboration of a "new" solution. If a problem has not yet received a solution during the current problem-solving, the starting point for solution elaboration is the representation of the problem. Cognitively, this is the most complicated solution-development mode.

Elaboration of an "alternative" solution

For many specific problems, the designer knows of only one solution: he evokes it and evaluates it. If it is rejected and he thus has no other solution left in memory, an "alternative" solution has to be elaborated. The designer has been observed to adopt different approaches.

Two general ones, leading to several possible solution-development processes, will be presented: modifying the problem by generalizing it, and modifying the solution.

Problem-modification by generalization. Generalizing a problem amounts to setting aside some of its attributes and focusing on others. The designer constructs this more general representation of the problem in the hope that it will activate an appropriate problem/solution.

Two possible forms of solution elaboration following this general approach will be presented. Both lead to a solution elaboration in two steps, going up and then down again in the problem/solution-abstraction hierarchy.

Generalizing the problem in order to activate a more abstract problem representation. This solution elaboration consists in first elaborating an alternative solution at a higher abstraction level (the new problem representation) and then elaborating a solution to this new problem at a lower abstraction level. That is, transforming a specific solution (Sol-s1) into an abstract solution (Sol-a) is followed by specifying Sol-a into a specific solution (Sol-s2) different from Sol-s1.

Generalizing the problem in order to activate a problem/solution schema. This second two-step solution-elaboration procedure applies to the elaboration of an alternative solution as well as to the elaboration of a "first," "new" solution to a problem.

If the representation of a specific problem Pb-s1 does not evoke an appropriate solution, the designer may see if a more abstract problem-representation activates a problem/solution schema with a default value for the attribute corresponding to Pb-s1. If so, this default value (or a specification of it) may then be proposed as a solution.

Solution-modification heuristics. Faced with a problem for which he knows of only one solution which he has evoked and rejected, a designer is supposed to start alternative-solution elaboration by modifying his problem representation (see above). If neither directly (via a schema), nor indirectly (via a more abstract problem representation) an appropriate solution can be elaborated, several heuristics are available for modifying the rejected solution.

Example. "Invert the value" of the "critical" problem/solution attribute, that is, of the attribute having led to rejection of the solution.

Elaboration of a "new" solution

Alternative-solution elaboration is only applicable when one has a solution (the evaluation of which has led to rejection). This solution and the result of the evaluation constrain the

alternative-solution elaboration. But without such a starting point, and without another solution stored as such in memory, a new solution has to be elaborated "from scratch."

"Design from scratch." The global design process -even of a completely "new" artefact- never proceeds from scratch: a difficult design project is never confided to a complete novice, and as soon as a person has some experience, he cannot proceed from scratch. However, solving sub-problems may proceed from scratch.

"Elaboration from scratch" remains to be defined. In this text it is considered to apply to solution development from material which is only analogous or similar, or even without any clear link to known, past analogous designs. This solution-development mode is probably the most difficult for the designer.

Various processes and strategies may be supposed to be used to this end. Those presented below were observed on the antenna project.

Brainstorming. This term, generally used for a group activity, here refers to an individually used very "weak" solution-"search" mechanism: transitions from one mnesic entity to another are not a function of rules (as in deductive reasoning), nor are they oriented by a search based on similarities (as in analogical reasoning). The activation is not "oriented" in the sense of "consciously" directed. Search is only "guided" by the links that may exist between mnesic entities, and that lead from activated "targets" to other mnesic entities, which may activate, in their turn, still other mnesic entities. The "targets" are those entities that have been activated as a result of having been processed by the designer in his construction and further processing of the problem representation. The only way in which a person may "guide" brainstorming is in his elaboration of the problem representation. The solution the designer comes up with, or the "idea" he "thinks about," is (or are) the newly activated entity (or entities) which exceed(s) a threshold.

Brainstorming thus is the exploitation of the "natural" memory access mechanism, i.e., spreading activation. The observer suspects brainstorming to take place when the designer hesitates, falls silent, and/or comes up with an "idea" or "solution proposal" after a certain

silence. In order to decide that brainstorming has been used -possibly accompanied by other processes- another condition must be satisfied: the observer must have a hypothesis concerning the transition path from the target(s) supposed to have been activated by the problem representation to the solution the designer proposes.

Like the other explanations proposed, such hypotheses remain of course to be verified. With these reservations, the problem-solving path for the antenna-design problem showed a frequent use of brainstorming.

Analogy-directed memory access. This process is an "oriented" form of activation. Even if it is not (yet) clear how this "orientation" from targets to analogous sources may proceed, the hypothesis is formulated that the sources are "searched for" through links between target and source mental entities (see Visser, 1991a, for a detailed analysis of this use of analogy)

In the two other studies presented in this paper, analogical relationships were exploited all along the design process. On the antenna project, their use was observed to occur especially during conceptual solution development.

Example. In a discussion with other designers, the observed designer and his colleagues developed "unfurling principles" for future antennas. They proposed conceptual solutions such as "umbrella," and other "folding" objects, like "folding photo screen," "folding butterfly net," and "folding sun hat." The designers thought of taking these objects into pieces in order to find possible ideas for the corresponding material solutions.

All proposed objects satisfied an important constraint that existed on the unfurling antenna, that is, they all had a trigger mechanism leading to such an unfurling that the resulting object constituted a rigid surface.

DISCUSSION

After a short recall of the main results of the three studies presented in this paper, their possible relevance for A.I. will be discussed.

Main results of the presented studies

The main contribution of the study on the global organization of the specification activity was the detailed analysis of how its opportunistic character was realized. Other studies have characterized design activity as opportunistic. Two important new ideas were developed in the present study. The first one is that pre-existing plans may be considered as only one of the knowledge structures used by the control to guide the activity (see Visser, 1991b). That is, at the control level of his activity, an expert designer has, next to a global plan making action proposals at each step in the design process, other knowledge structures which propose alternatives to the plan-proposed actions. Five processes leading to such deviation-action proposals were presented.

The second idea is that an important factor underlying the opportunistic character of the actual design activity is on the action-management level. Only as long as they are cognitively cost-effective, the control selects for execution actions proposed by the global plan ("planned" actions). As soon as other actions are more interesting from a cognitive-cost viewpoint, deviations from the global plan occur in favour of these actions ("deviation" actions) (see Visser, 1991b).

The analysis of design strategies focused on the identification of local strategies specific for working on real, complex design projects. Trying to characterize an activity in terms of top-down or bottom-up and depth-first or breadth-first strategies is more or less successful according to the level at which the analysis underlying this characterization takes place. At a high, abstract level, the observed programmer's activity could be described as following a top-down strategy: the program was decomposed into modules and coding was planned to follow this decomposition. As soon as the activity is analyzed at the local level, that is, at the level of the actual design actions, deviations from this plan are observed and the activity is no

longer top-down: for example, semantic relationships between the mental representations of different design components are exploited, leading to deviations similar to those described in the section on the organization of the design activity.

With respect to the local strategies, some strategies already known to be at work in "designing-in-the small" were observed, for example, simulation. Other strategies seem indeed to be characteristic of designing in a work context, especially the re-use of program components and the consideration of users of the system. Neither has generally been noticed in previous design studies. An explanation of this absence might be that their implementation depends on conditions only realized in real work situations.

Actual re-use of existing components was observed, but the programmer also created facilities for possible re-use. The observed homogenizing of inter-program variable numbering is an example of this anticipation oriented towards future re-use. Consequences of this result will be developed and discussed below.

The local strategies were implemented in combination, mostly of two. So, simulating the operation of the machine tool while considering (a particular class of) users of the system might make the programmer think of elements to be included in the design.

The analysis made in the third study was more descriptive. There were no underlying hypotheses to examine or claims to refute. Several results were especially interesting. So were the frequent use of "brainstorming" in all solution development stages, and the importance of analogy-directed memory access during conceptual solution development. The study has inspired many questions and some hypotheses:

• When and how do problem/solution schemata develop? How does their appearance modify the associations between particular problems and solutions?

• The problem of the "scope" of a solution's activation. When a problem/solution is activated, what is it that is activated? Its "label" is activated, but not only that. Attributes and values are activated, but not all of them. It surely depends on the problem representation the designer constructs, but the problem remains open to question.

• What about the proposed "brainstorming?" Is there a process that may be identified as such? Is it rather a series of activations: in that case, what is their unity?

• What about "classical" problem-solving processes like deductive reasoning? Why were they not observed? Do they not play a role in other than routine-design activities?

Possible relevance for A.I.

In the introduction, we claimed that the results of studies like those presented above are relevant for the design of knowledge-based systems. This claim will now be defended through the examples of results on decomposition and re-use, but the other results could have been used as well. Three examples will be presented. The design of support tools for strategies such as simulation is a complex task and could take advantage of empirical data on the representations constructed by designers and the way in which they are used. It would also be useful to assist the switching between the various viewpoints which designers take on the design problem/solution state. Finally, given the problems met by designers due to memory limitations, displays for helping the management of working memory could be helpful, such as facilities for:

• parallel presentation of intra- or inter-level information;

• presentation of the constraints on the solution order;

• maintaining a trace of postponed sub-problems needing backtracking.

Assistance tools for the management of memory load could also support simulation, given the frequent need for a designer involved in simulation to hold simultaneously several variables in mind. More ideas on assistance and references to other studies may be found in Ullman, Staufer & Dietterich (1987), Visser & Hoc (1990) and Whitefield (1986).

Decomposition. If the design activity is opportunistically organized, an assistance system which supposes -and therefore imposes- a hierarchically structured design process will at the very least constrain the designer and will probably even handicap him (see Visser & Hoc, 1990). Assistance tools should be compatible with the actual activity. In the present case, such tools should allow the design -that is, the solution under development- to be

abandoned at a certain level when the designer prefers to process solution elements at another level. The system should not only allow the design to be resumed later at the abandoned level, but it could assist this resumption by keeping a trace of such abandonments. This implies that the system knows the structure of the final design. The system should not, however, impose such a resumption. The engineer we observed used a hierarchically structured plan, to which he returned after deviation actions. Not all designers necessarily refer to such a plan, especially if the (final) design has a less constrained structure than the specifications constructed by the observed engineer.

Re-use. Re-use is an approach advocated by many computer scientists working on design environments in software and other design domains (see IEEE Software, Special issue on reusability, 1987; Artificial Intelligence, special issue on Machine Learning, 1989). Arguments in support of the approach are based on "introspection" and "common sense" considerations (economy etc.), not on results from cognitive studies on designers. Besides, if psychological studies exist on the design activity in general, empirical research on how people actually re-use design components is almost non-existent.

Systems allowing for re-use or based on re-use are being developed. However, as formulated by a computer scientist building systems supporting redesign, "The disadvantage [of this type of complex systems] is that [reusable] building blocks are useless unless the designer knows that they are available and how the right one can be found. … Several cognitive problems prevent users from successfully exploiting their function-rich systems." (Fischer, 1987, p. 61)

In order to offer users really helpful systems, these "cognitive problems" must be identified together with the strategies actually used by designers to solve them.

REFERENCES

Anderson, J. R. (1983). The architecture of cognition. Cambridge, MA: Harvard University Press.

Artificial Intelligence, special issue on Machine Learning (1989) **40** (2).

Bisseret, A., Figeac-Létang, C., & Falzon, P. (1988). Modeling opportunistic reasonings: the cognitive activity of traffic signal setting technicians (Research Report N° 898). Rocquencourt: INRIA.

Darses, F. (1990). Constraints in design: Towards a methodology of psychological analysis based on AI formalisms. In D. Diaper, G. Cockton, D. Gilmore & B. Shackel (Eds.), Human-computer interaction - INTERACT '90. Amsterdam: North-Holland.

Eastman C. M. (1970). On the analysis of intuitive design processes. In G. Moore (Ed.), Emerging methods in environmental design and planning. Cambridge, MA: MIT Press.

Ericsson, K. A. & Simon, H. A. (1980). Verbal reports as data. Psychological Review, 87, 215-251.

Ericsson, K. A., & Simon, H. A. (1984). Protocol analysis. Verbal reports as data. Cambridge, Mass.: MIT Press.

Fischer, G. (1987). Cognitive view of reuse and redesign. IEEE Software **4**: 60-72.

Guindon, R., Krasner, H., & Curtis, B. (1987). Breakdowns and processes during the early activities of software design by professionals. In G. Olson, S. Sheppard & E. Soloway (Eds.), Empirical Studies of Programmers: Second Workshop. Norwood, N.J.: Ablex.

Hayes-Roth, B. & Hayes-Roth, F. (1979). A cognitive model of planning. Cognitive Science **3**: 275-310.

Hayes, J. R., & Flower, L. S. (1980). Identifying the organization of writing processes. In L. W. Gregg, & E. R. Steinberg (Eds.), Cognitive processes in writing. Hillsdale, N.J.: Erlbaum.

Hoc, J.M., Green,T., Samurçay, R., & Gilmore, D., (Eds.). (1990). Psychology of programming. London: Academic Press.

IEEE Software, Special issue on reusability (1987) **4** (4).

Kant, E. (1985). Understanding and automating algorithm design. IEEE Transactions on Software Engineering **SE-11**: 1361-1374.

Pennington, N. & Grabowski, B. (1990). The tasks of programming. In J.M. Hoc, T. Green, R. Samurçay & D. Gilmore (Eds.), Psychology of programming. London: Academic Press.

Ullman, D., Staufer, L. A., & Dietterich, T. G. (1987). Toward expert CAD. Computers in Mechanical Engineering **6**: 56-70.

Visser, W. (1987). Strategies in programming programmable controllers: a field study on a professional programmer. In G. Olson, S. Sheppard & E. Soloway (Eds.), Empirical Studies of Programmers: Second Workshop. Norwood, N.J.: Ablex. Also accessible on http://hal.inria.fr/hal-00641376/en/

Visser, W. (1988). Giving up a hierarchical plan in a design activity (Research Report N° 814). Rocquencourt: INRIA. Also accessible on http://hal.inria.fr/inria-00075737/en/

Visser, W. (1990a). Evocation and elaboration of solutions: Different types of problem-solving actions. Actes du colloque scientifique COGNITIVA 90. Paris: AFCET. Also accessible on http://hal.inria.fr/inria-00000165/en/

Visser, W. (1990b). More or less following a plan during design: opportunistic deviations in specification. International Journal of Man-Machine Studies. Special issue: What programmers know **33**: 247-278. Also accessible on http://hal.inria.fr/inria-00633544/en/

Visser, W. (1991a). Analogical reasoning: different applications in design problem solving. Paper presented at the Symposium on AI, Reasoning & Creativity, Lamington National Park, Brisbane, Queensland (Australia), 20-23 August 1991.

Visser, W. (1991b). Planning in routine design. Some counterintuitive data from empirical studies. In J. S. Gero & F. Sudweeks (Eds.), Preprints of "Artificial Intelligence in Design", Workshop of the Twelfth International Joint Conference on Artificial Intelligence, Sydney (Australia), 25 August 1991.

Visser, W., & Hoc, J.M. (1990). Expert software design strategies. In J.M. Hoc, T. Green, R. Samurçay & D. Gilmore (Eds.), Psychology of programming. London: Academic

Press. . This chapter is also accessible in Dr Alan Blackwell's Course material 2010–11 for "Usability of Programming Languages" http://www.cl.cam.ac.uk/teaching/1011/R201/ with the following "Note on copyright: This book is currently not available in print. A selection of chapters has been provided, with the permission of the editors, for non-commercial educational and research use only. Members of the Psychology of Programming Interest Group (PPIG) continue to conduct research as outlined here, and it is intended that specific chapters will be replaced with updated versions when individual authors are able to provide them. Volunteer assistance with OCR of these chapters, to assist those authors who no longer have their original manuscripts, would be appreciated."

Whitefield, A. (1986). An analysis and comparison of knowledge use in designing with and without CAD. In A. Smith (Ed.), Knowledge engineering and computer modelling in CAD. Proceedings of CAD86. Seventh International Conference on the Computer as a Design Tool. London: Butterworths.