DOCUMENTATION AS PART OF DESIGN: EXPLORATORY FIELD STUDIES

Jean-François Rouet, Catherine Deleuze-Dordron and André Bisseret

National Institute for Research on Informatics and Automation
46 avenue Félix Viallet, 38031 Grenoble Cedex, France.
Phone (+33) 76.57.47.76; Fax (+33) 76.67.46.95; e-mail rouet@isis.imag.fr

KEY WORDS: software design, design with reuse, documentation, problem solving, support environments, design expertise.

ABSTRACT: We describe a set of exploratory field studies in which we examined expert software designers' documentation strategies. Preliminary studies showed that there is no single agreed upon documentation methodology. However, content analysis of the literature as well as expert interviews indicted that inserted comments fulfil several explanatory functions. A case study of documentation as part of the testing of a prototype toolset indicated that design with reuse requires a great deal of knowledge about the application domain, previous applications and reuse methodology in general. Moreover, documentation tools were extensively used during design. Content analyses and expert interviews highlighted the need for a close match between the design of those tools and the designers' actual strategies. Some implications concerning the design of support environments are suggested.

1. INTRODUCTION

Designers (e.g., architects, engineers or computer scientists) rely extensively on complex documentation systems. In software design the issue of producing quality documentation has become more and more critical as software engineering methodologies are increasingly sophisticated and demanding. In particular, current efforts to improve software design quality and productivity through component reuse highlight the importance of documentation and retrieval schemes (Frakes et al., 1991).

In order to facilitate the designers' task, and to improve the quality of software artefacts, design support environments are being proposed. Support environments typically include component libraries as well as tool kits to design, store, retrieve and edit software components. In these environments a precise, accurate and relevant documentation of the components (e.g., interface, provided functionality, specific properties) is a key factor of reusability (Lehner, 1993). However, the design of usable and efficient documentation tools as part of a support environment requires a comprehensive model of the design activity, which is yet to be proposed.

From the point of view of cognitive psychology, software design is considered a complex problem solving activity (Curtis, 1991; Soloway et al., 1982). According to the information processing theory (Newell & Simon, 1972), solving a problem involves reaching a goal (or "final state") through a series of intermediate steps. Software design problems are known to be "ill-structured", i.e., there may be several acceptable solutions for a given problem. Moreover, several strategies (i.e., definition and organization of intermediate steps) may lead to a given solution.

Hayes-Roth and Hayes-Roth (1979) have proposed the notion of "opportunistic planning" to describe the organization of complex cognitive activities. According to their model, designers may initially follow a hierarchical framework, but they may depart from it *en route* based on characteristics of the specific problem situation (see also Guindon, 1990; Visser, 1987).

Empirical studies of programming have indicated that software designers rarely use a strictly top-down strategy. Instead, they display episodes of opportunistic behavior. Although a hierarchical top-down approach may be applicable to simple problems, there is evidence that experts will depart from this approach when difficulties arise (Guindon, 1990; Hoc, 1988).

Because design is such a complex, multilayered activity, designers need to keep track of their own processes through notes, graphical representations, schemas, etc. In addition to their need for "private" documentation, designers have to communicate their solutions in explicit forms, and hence they have to provide "public" documentation for the storage, maintenance, and reuse of the software.

Even though a respectable amount of theoretical and empirical work has been devoted to the psychology of programming, the process of software documentation is still obscure. There is not yet a general cognitive model that may help answer questions such as: What is the exact role of documentation in the design process? What types of information are needed, and at which phases of the design process? How can documentation help designers identify and reuse existing software components? In an attempt to address these questions, we undertook empirical field studies of the cognitive and human factors issues involved in designing and using "internal" documentation, i.e. documents issued *as part of* the design and reuse of software components.

In the next part of this paper we summarize a series of preliminary field studies aimed at exploring documentation issues in SW engineering literature and in expert designers. Then we outline a case study of a design with reuse support environment currently under development. We focus on the ergonomics and cognitive aspects of using the environment, and we draw a few implications for the design of documentation tools in this type of environment.

2. PRELIMINARY FIELD STUDIES

2.1. Documentation Prescriptions in Software Literature

As a first step we looked for documentation prescriptions in the software engineering literature (Deleuze-Dordron, 1993).

We examined a series of six major software engineering manuals in order to identify principles or guidelines concerning what, when and how to document software. Information on this topic was scarce in most of the books studied. In only one case we found a tentative typology of inserted comments. Although most manuals gave little consideration to the problem of documentation, most of them included examples of programs or algorithms with a large number of inserted comments. Thus, the production of inserted comments seems to correspond to implicit common sense heuristics rather than explicit principles.

We also examined two sets of design guidelines, one from a team of scientists, the other from a software engineering company. Most guidelines were rather general (e.g., "*avoid redundancy*"). Moreover some guidelines relied on unsupported psychological hypotheses, e.g., "*redundancy increases the reader's processing load*". Very few were explicitly related to some kind of model of the design process.

2.2. Roles and Uses of Documentation: Analyzing Expert Representations

The purpose of our second exploratory study was to gather information on expert designers' representations and beliefs about the use of commenting in software design. We interviewed five expert designers (four computer science faculty members and one designer from a SW engineering company) according to a semi-directed interview scheme.

Most of the experts agreed that in general programs are poorly documented. They often cited program comprehension and/or component reuse as situations where the need for documentation is most critical. However the experts acknowledged that there is so far no agreed upon method to devise appropriate comments. Instead, commenting seems to vary according to each expert's strategies, habits and style. Nevertheless, four categories of information consistently emerged from the interviews: Purpose, time and position of comments, as well as constraints on comment production.

Purpose of commenting. According to the experts, comments may convey information about the design *process* and the design *product.* First, comments may provide explanations about the solution or the method used to solve a problem. This information can be useful for the designer (it serves as an "external working memory") and for other readers. Second, object names and inserted comments play a major role in understanding a program. Names and comments should explain the meaning of an object or an action. They should also mention its exceptions, limits and context of use.

Time of commenting. Comments may be issued before, during and after program design. Thus, commenting can be part of the reasoning itself, i.e., a means to plan or to make explicit intermediate steps of the problem-solving activity.

Position of comments. Comments may contain information about the design of an application (high-level comments) or about its implementation (low level comments). In addition, comments can be located at the beginning of a source file, at the beginning of functional units and close to particular statements within units.

Constraints on comment production. The experts also pointed out external (e.g., time and cost) and internal constraints (e.g., type of language used) on the production of comments.

2.3. Discussion

The objective of our preliminary studies was to examine the role and use of documentation in software design. Our review of the literature and guidelines elicited several functions of commenting. The interviews indicated that expert designers use several criteria in defining what a good documentation should look like. Together these findings indicate that internal documentation plays a role as part of the design process itself. They may be produced during the design process, and may be addressed to the designer herself or to other readers.

There are intriguing similarities between program commenting and note taking during learning activities (Kiewra, 1989). Notetaking facilitates both the understanding of a piece of text and the retrieval of information from that text. Similarly, commenting seems to assist the design process, in addition to improving the readability of a piece of software.

3. A CASE STUDY OF DOCUMENTATION IN DESIGN WITH REUSE

3.1. Presentation of the Study

As part of the EC project ESPRIT-SCALE, we were involved in the empirical evaluation of a design with reuse support environment under development (see D'Alessandro et al., 1993; 1994).

The environment includes a library of reusable components and a set of tools to extract, adapt and document the components. The system is based on the HOOD methodology (Hierarchical Object Oriented Design, see HOOD Technical Group, 1993). The environment includes a tool to document components during the reuse process (reuse notebook). The reuse notebook includes several free text attributes aimed at storing information about the decision to reuse a component and how reuse is to be conducted (modifications, etc.).

At the time of the present study a first prototype was being tested based on a series of applications in the domain of hard real-time executives. A library had been developed based on a previous application, and was to be used to develop a new application. The testing activity involved two types of design experts: *Application experts*, whose role was to develop the target application, with little previous experience of the reuse environment; *Toolset experts*, who had participated in the development of the environment and whose role was to assist the application experts.

This situation was especially interesting in our perspective since it allowed to study the types of human expertise involved in design with reuse (D/R) as well as the production and use of documentation as part of this activity. Our method primarily consisted in collecting various types of data:

- Observation of a D/R session including the recording of dialogs between experts and the report of critical incidents.

- Interviews with the toolset and application experts, at the begining, during and after the design phase.

- Analysis of technical documents about the environment (tools, design guidelines, user manual) and outputs from the activity (e.g., internal documents).

3.2. Results

Only the main outcomes will be reported in the present paper. (see Rouet & Deleuze-Dordron, 1994, for more details). First we present an analysis of the expertise involved in D/R; second, we focus on the use of documentation tools as part of the D/R activity.

3.2.1. Types of expertise involved in D/R.

The field testing involved two major phases. First, a library of reusable components (RCs) was populated using a source application; second, the library was used as part of the development of the new application. For each phase we identified the expertise involved and how the prescribed method and toolset were exploited by the designers.

Populating the library. This activity involved identifying RC candidates in the source application and preparing them for inclusion in the library.

Based on our observations of the designers' activities, we found that identifying RC candidates requires at least three types of expertise on the part of the designer.

First, the designer must be an application domain expert in order to understand what functionalities might be similar across applications, which will drive the search for RCs.

Next, the designer must possess detailed knowledge of the source application, so as to understand the particular implementation of reusable functionalities.

Finally, the designer must be an expert in reuse strategy. "Shaping" RCs involves making a number of decisions, e.g., choosing an optimal abstraction level. These decisions are based on various rules that are not totally explicit in the proposed reuse model. In certain cases the selection of RCs might be intuitive or arbitrary (see Krueger, 1989).

Using the library. The use of the library during the development of the target application involved three main tasks: Identifying and selecting RCs previously stored in the library; (if necessary) modifying the selected RCs; and including the selected RCs into the new application.

Again, several types of expertise are required when performing those tasks: First, the designer must have a thorough knowledge of the application domain. S/he must know the requirements of the target application, its similarities and differences compared to other applications in the same domain. We found indications that some knowledge of the source application may also be needed. In the present field study, a single application was used to populate the library. Furthermore, both the source and target applications were developed by the same experts, who were also involved in the extraction of RCs. Thus, in this case not only were the designers domain experts, but they also had much previous knowledge of the specific library components. Whether reusers with no previous knowledge of the source application would have managed to identify and select RCs in the proposed environment is still an open issue.

We also observed that the application experts needed paper documentation about the source application at the time of the first D/R session. This is a further indication of the importance of source application knowledge in the D/R process. Moreover, it suggests that the RC library did not include all the needed information.

Applying the reuse model requires a good deal of skill in the HOOD design method, and in the use of the support environment. In fact, during the first D/R session the application team experienced several types of difficulties. First, the designers needed assistance when trying to locate and use the toolset functionalities, which indicates that operating the toolset is not a trivial aspect of the activity. There were a few attempts to perform illegal or inappropriate operations, suggesting possible misconceptions of the "activity model" implemented in the toolset. Finally, the reuse notebook was hardly used spontaneously by designers, who tended to use paper notes instead. For instance one expert indicated that she liked to sketch graphical HOOD representations on paper before drawing them on the computer screen.

It is important to point out that the observed session was the very first one, which may account for some of the observed difficulties, despite prior training efforts. Nevertheless, the fact that a toolset expert was helpful to the application experts indicates that (a) toolset expertise plays an important role in the successful application of the reuse model and (b) such expertise is not readily available to the users, even though they may be expert designers.

3.2.2. Use of documentation tools. Two types of data were used: First, we performed a content analysis of the documents produced as part of the design; Second, we asked two experts to comment on a sample of those documents.

Content analysis: The reuse notebook included two free text attributes aimed at documenting the reuse process. The first attribute ("*Why included?*") concerned each component reused as part of the design of a new parent component (local attribute). It allowed the designer to state the reasons for reusing the component. The second attribute ("*Reuse planning*") concerned the parent component as a whole (global attribute). It allowed the designer to state his or her plans concerning the reuse process within the design of the parent component.

We found that the local attributes were seldom used. Most often, the designer made rather trivial statements, e.g. "*[the reused component] provides functionalities which have to be also implemented in the system to design*". The global attribute was used to store various types of information. Some statements reflected an actual planning activity (e.g. "*before reusing the object x, it has to be adjusted to the new environment*"). However, the statements concerned the children (reused) components as well as the parent component. Thus there seemed to be a need for planning information at the global AND local level.

We also noticed that the application experts tended to use tools external to the reuse system to store natural language information. In particular, they wrote statements about the actual decisions made during the design with reuse process. Thus, the designers needed a "logbook" of design decisions in addition to the "planning" attribute.

Expert interviews: We conducted parallel interviews with an application expert and a toolset expert. The interview protocols involved a series of open-ended questions based on hard copies of documentation screens. The toolset expert provided explanations for the current design of the documentation tools based on the design method implemented in the toolset. However, the application expert stated several difficulties with the toolset. For instance, she mentioned that the "reuse planning" attribute should be implemented for local as well as global objects; she also stressed the need for a text attribute where final decisions could be stored. The expert also justified her use of an external tool by the lack of such a tool in the reuse system.

3.3. Discussion

The case study illustrated several important characteristics of the design with reuse activity. Obviously, reusers have to be experts in the application area. However, our data indicated that they also rely on two other types of knowledge: knowledge of previous applications used to populate the library; knowledge of rules and heuristics on when and how to reuse components. Our observation of the documentation tools showed that reusers need to store natural language information about the design process at all levels of the application architecture.

In addition, our study showed the importance of providing appropriate labels for free text attributes. In fact, the reusers could have used the provided attributes to store any type of information they wished. However, they did not do so. Instead, they tried to comply with the "directions" suggested by the attribute names. However, this proved to be difficult or sometimes impossible. As a consequence the reusers either misused the attributes or looked for other locations in the system in order to store those types of information which were not explicitly supported.

Based on these observations, we suggested to replace the two reuse documentation attributes with a "local reuse logbook" (for each low-level object) and a "global reuse logbook" (for the parent object).

4. CONCLUSIONS

The main purpose of the studies reported in this paper was to explore the production and use of natural language documentation as part of the software design process. A better knowledge of the designers' documentation strategies may help define guidelines for the integration of documentation tools into design support environments.

Our preliminary studies suggest that natural language information can serve various functions in the design of a piece of software. Comments may specify the use of a variable or object; make explicit the purpose of a procedure; explain the functioning of an algorithm. Moreover, comments seem to be issued as part of the designer's reasoning process.

Our case study of design with reuse confirmed these preliminary findings. We found that designers rely on several types of knowledge, including memory of past applications in the domain as well as rules and heuristics concerning when and how to reuse. They made extensive usage of the documentation tools provided as part of the support environment. However, we found discrepancies between the proposed labels and level of abstraction of the free text attributes and their actual usage. For instance, the designers did not have much to say about the reasons to include a reusable component, but they would have needed an attribute to state the reuse planning of the reused components. These results illustrate the need to pay close attention to the designers' actual strategies when designing a support toolset.

A serious problem for the development of design with reuse support environments is the need for contextual information, i.e., information about the context (application, purpose) in which the component was originally designed. We suggest that this problem is related to the cognitive constraints of design activities. As pointed out by Détienne and Bisseret (1993), software design problems are often ill-structured, in that several solutions may be used to solve a problem. The designer's task is to select the most appropriate solution. Previous experiences in the same application domain play an important role in this decision process. For instance, the re-examination of previous applications (not only components) can help decide on whether or not to reuse existing components, and at what level of abstraction reuse might take place (see Kruger, 1989).

Another problem is that effective usage of a support environment requires the expert to learn the functionalities and the interface. When confronted to a new set of tools, even computer science specialists have to go through a learning phase during which they explore, make mistakes and spend extra-time performing their task. Although toolset expertise can be acquired through hands-on experience, the consequences in terms of reusers' training needs should not be overlooked.

Finally, it should be pointed out that the studies reported here are only preliminary steps toward a general cognitive model of documentation processes as part of design activities. Whenever possible, we have attempted to draw implications of our observations for the design of interactive environments. However, more field studies will be necessary in order to provide general recommendations on how to provide effective support for design activities.

REFERENCES

Curtis, B. (1991). Cognitive issues in reusing software artefacts. In T. Biggerstaff & A.J. Perlis (Eds.), *Software Reusability, v. II*. New York, NY: ACM.

D'Alessandro, M., Lachini, P.L., Martelli, A. "The Generic Reusable Component: An approach to Reuse 00 Designs" *Proc. of Second International Workshop on Software Reusability*, Lucca (Italy), March 24-26 1993. IEEE Computer Society Press.

D'Alessandro, M., Martelli, A. "ReuseNICE: A toolset to nicely support reuse". *Proc. of the XIV International Conference of the Chilean Computer Science Society* - Conception, Chile October 31 - November 4, 1994.

Deleuze-Dordron, C. (1993). *Analyse de l'activité de documentation de programmes: première exploration.* Mémoire pour le DEA de Sciences Cognitives, Grenoble: INPG, 1993.

Détienne, F. & Bisseret, A. (1993). *A study of ergonomic and cognitive aspects related to the man-machine interface of process support environments.* SCALE Deliverable D2.1.2-1

Frakes, W.B., Biggerstaff, T., Matsamura, K., Prieto-Diaz R., & Schaefer, W. (1991). Software Reuse: Is it delivering? *Procs 13th Conference on Software engineering*. Austin, TX, May 13-17.

Guindon, R. (1990). Designing the Design process : Exploiting Opportunistic Thoughts. *Human Computer Interaction*, **5**, 305-344.

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

Hoc, J.-M. (1988). *Cognitive psychology of planning*. London: Academic Press.

HOOD Technical Group (1993). *HOOD Reference Manual 3.1*. Paris: Masson

Kiewra, K.A. (1989). A review of note-taking: The encoding-storage paradigm and beyond. *Educational Psychology Review*, **1**, 147-172.

Lehner, F. (1993). Quality control in software documentation based on measurement of text comprehension and text comprehensibility. *Information Processing and Management*, **29**(5), 551-568.

Newell, A. & Simon, H. (1972). *Human problem solving*. Englewood cliffs, NJ: Prenctice Hall.

Rouet, J.-F. & Deleuze-Dordron, Catherine (1994). *Design with reuse and software documentation: Some cognitive and human factors issues.* INRIA Rhône-Alpes, unpublished manuscript.

Soloway, E., Ehrlich, K., Bonnar, J. & Greenspan, J. (1982). What do novice know about programming? In A. Badre & B. Schneiderman (Eds.), *Directions in Human Computer Interactions* (pp. 27-54). Norwood, NJ: Ablex Publishing Corp.

Thunem, S. & Sindre, G. (1992). Development with and for reuse. Guidelines from the REBOOT project. Procs ERCIM Workshop on methods and tools for software reuse (pp. 2-16). Heraklion, Greece: October 29-30.

Visser W. (1987). Giving up a hierarchical plan in a design activity. INRIA, tech. Rep. No 814.