Problem Acquisition in Software Analysis: A Preliminary Study

Stephen Fickas
Sharon Collins
Susan Olivier

CIS-TR-87-04
January 13, 1988

Department of Computer and Information Science
University of Oregon

Problem Acquisition in Software Analysis: A Preliminary Study

Stephen Fickas
Sharon Collins
Susan Olivier

Computer Science Department
University of Oregon
Eugene, Or. 97403

Abstract

We are interested in how a problem description can be acquired from a client, subsequently debugged, and finally turned into a formal software specification. In this paper we discuss a) the components of a proposed system for handling these processes, b) the results of preliminary protocol studies in support of the acquisition component of our system, and c) our prescription for further work in the area. We include an Appendix that contains pieces of our protocols that illustrate two important acquisition techniques that emerged from our studies, example generation and summarization.

1. Introduction

We are interested in automating the analysis phase of software development[1]. We are working on a system that attempts to acquire a problem description that includes requirements on the objects, actions, and constraints of the intended system, as well as descriptions of the time and labor resources available during development, and the constraints on the run-time environment. In this effort, we make the following assumption: the user may have only a vague idea of what he or she wants. This assumption runs counter to the view of acquisition as a process of *translating* or *rephrasing* user intent to formal or semi-formal representations. In our view, problem acquisition is not so much a translation process as an interactive *problem solving* process with both user and analyst involved in supplying parts to the final product. In particular, we are attempting to extend analysis into validation of intent. That is, we expect to take a perfectly valid (syntactically correct, consistent, unambiguous) description of a problem, and attempt to poke holes in it. Our analysis will be based on a model of what is common and desirable in the domain.

We have been testing our ideas in a system called KATE, described in detail in [4]. KATE is being built around the following components:

- A model of the domain of interest. This includes 1) the common objects, operations, and constraints of the domain, 2) the policies (nee requirements) of the domain, and 3) a model of the way the resulting system will be used by the users/environment.

- A problem acquisition component that extracts the details of a problem from a client.

- A critic that attempts to poke holes in the current problem description. Bugs that are of interest to the critic include syntactic errors such as missing inputs, dangling processes, type mismatches, as well as "intent errors" that violate KATE's view of typicality.

A demonstration system that illustrates the use of the domain model component in conjunction with the domain critic components has been built, and is described in [3]. Our next planned step is to include the problem acquisition component in the demonstration system. A prerequisite of this step is an understanding of the foundations of problem acquisition in software analysis. In this paper we discuss our attempts to build such a

[1]Our use of the term analysis is fairly broad, and includes problem acquisition, validation, and production of a formal software specification. It tends to encompass traditional notions of system and domain analysis, requirements analysis, and specification.

foundation.

2. What Makes a Good Software Analyst?

We'd like to be able to report that we have a satisfying answer to this section title. We do not. However, we have begun to glimpse at least a few of the skills of expert software analysts, and describe our observations in more detail in this section.

2.1. Initial Hypotheses

Over the last four years, we have informally observed the software projects in our two quarter, Senior Project course. This has been a good testbed: we have been able to view not only the analysis process, but each of the subsequent post-analysis phases of development. In particular, we have been able to interview clients[2] well after the software has been in use, and find their level of satisfaction with its functionality, interface, and general performance. Using this information, we have sometimes been able to trace client complaints back to analysis bugs[3]. This has lead us to hypothesize the following: that a finer grain classification of analysis bugs than the traditional *incomplete, inconsistent, ambiguous* may be possible; that certain analysis techniques do better at avoiding certain classes of analysis bugs; conversely, that certain types of analysis deficiencies lead to certain analysis bugs. In particular, we are interested in the following two hypotheses:

Hypothesis 1: A productive software analysis technique is *example generation.* We have observed that the more examples that can be generated showing the system in use, the better the chance of finding the special-case events and states that must be represented. The best analysts among our students were the ones able to find an adequate level of representation by setting up hypothetical situations for a client to work through.

Hypothesis 2: A counter-productive software analysis technique is a belief that *the customer is always right.* Student analysts who relied on the client knowing what was possible and what was required generally produced only marginally acceptable systems. While students often held spirited discussions on what was deliverable in a given time frame, they rarely questioned the client's statements of need, acting more as Santa jotting down a wish list. In our observation, this is a critical factor in eventual client dissatisfaction

[2]To add realism to the course, students are encouraged to seek out clients from the local university and business communities with real software needs.

[3]This is not to imply that we have done rigorous analysis of customer complaints as they relate to analysis deficiencies. To date, our observations have been quite informal. However, we see much to be gained by doing a more formal study. Along these lines, we have begun to systematically record the development of a moderately large database system for a county government. Eventually this will include all group meetings dealing with the analysis, design, implementation, and acceptance testing of the system.

with delivered code.

Because of the inherent limitations in observing inexperienced student analysts in a largely academic setting, we devised a more realistic situation where we could observe experienced analysts working on larger projects, and keep both an audio and video record of the process. The next section discusses our work in this area.

2.2. Observing Experienced Software Analysts

After observing our students, we had two questions: 1) was example generation a key and observable technique of experienced, working analysts, and 2) did experienced analysts, when compared to student analysts, take a more critical view when interviewing a client?

Beyond these two questions, we were interested in gaining insight into a model of control for problem acquisition and analysis that might be used as a control model for KATE.

Our approach to gathering problem solving protocols was a rather catch-as-catch-can process. While we would have preferred a more structured study taken over a set of analysts, the logistics of finding experienced analysts in a particular domain proved a problem[4].

Below, we summarize the protocol sessions that were run.

(1) A library-systems analyst with 17 years experience interviewed two clients, one a high school librarian who was interested in automating her library, and one a high school administrator interested in a centralized, automated library system for her district. Two 1 hour protocols were generated. We will refer to this as protocol 1.

(2) A library-automation expert with 5 years experience gave a critique of an informal problem description used in a recent workshop on software specification and design techniques. One 1 hour protocol was generated. We will refer to this as protocol 2.

(3) A hardware/software analyst with 5 years experience interviewed two secretaries in an academic department about the need for an MIS to handle graduate admissions. One 1.5 hour protocol was generated. We will refer to this as protocol 3.

[4]This, and other aspects of our protocols that tend to weaken our findings are discussed in section 2.6.

In both protocols 1 and 3, clients were considering a computer system, and were willing to talk with an analyst about their requirements.

While protocol 2 is not directly related to problem acquisition, we include discussion of it because of the interesting ways the analyst/critic used examples to support her points. More on this below.

We have done a preliminary analysis of the three protocols, and summarize our findings in sections 2.3 - 2.5. Summaries for protocols 1 and 3 include what we learned about 1) a general dialog control model, 2) more specific software analysis techniques that we observed, and 3) the support the protocol gave to the two hypotheses generated from our work with student analysts. Discussion of protocol 2 is more general in nature.

Notation: a reference to **Ai** in the following sections refers to a portion of a protocol in Appendix A that tends to illustrate or support the point.

2.3. Findings from Protocol 1

General dialog control. The interview structure in both sessions was generally the same. The analyst first established his credentials and asked some context-setting questions. He next asked a set of stock questions that required the client to provide statistics on her environment (e.g., size of book collection, average transactions per day, size of staff, budget). This whole process lasted a surprisingly short time.

The analyst next came up with a rough solution. In both cases he suggested a PC-based system.

Finally, he spent the remainder of each session convincing the client that the solution would work, and exploring how the system would fit with the non-technical aspects of the client's environment, e.g., acceptance by higher-ups, acceptance by current library staff, funding problems. This took up the majority of each session.

Acquisition and analysis techniques. Looking in more detail, we found the following types of behavior occurring in each session:

- **Tutoring:** the analyst had a broader and deeper knowledge of automation possibilities than either librarian/client. He would often tutor the client on what was possible (**A3**).

- **Automation Impacts:** the analyst explored the impact of installing his proposed solution in the client's environment (**A2, A6, A8, A9**).

- **Concern Mitigation:** each client raised concerns during their session about the analyst's proposed solution. The analyst used various means to show how these concerns would be addressed by his solution (**A3, A7**).

As one can see from the above list (and by studying the amount of time devoted to each in the recorded session), actual information gathering and problem acquisition was at a minimum. The majority of issues centered on how to get the analyst's proposed solution in place at the client's site, and convincing the client that it was a good solution. In each session, the analyst did most of the talking. Further, all information was gathered under direct questioning by the analyst; the client was not given the opportunity to give more than short answers.

Support of hypotheses. It seems clear that the analyst was not working under the assumption that the customer is always right. On the contrary, one might view the results in the opposite vein: the customer is seldom right. He or she does not have a broad enough knowledge of automation alternatives to become actively involved in generating a solution. The client's main role then would seem to be to describe the details of his or her environment through direct questioning by the analyst (this can be contrasted with our observations of student analysts).

In the spirit of the above, this analyst used examples not so much to gather information, but in support of the goodness of the proposed solution and to mitigate a client's worries.

Finally, the analyst seemed to have a deep knowledge of the domain, and was not overly concerned about missing some hidden states in the client's environment. We could not see anything he missed, but remained suspicious of the little time spent exploring the usage patterns of the client's library. While it seems clear that the more knowledge an analyst has about a domain, the more usage knowledge he or she brings to the interview (and hence does not have to acquire anew), it also seems clear that each new problem will have some twists in it that will have to be explored. We saw little such exploration, and would predict that a system delivered on just the session that we observed would run into problems once installed (however, our comments in section 2.6 tend to weaken this argument).

Most of each session involved issues and techniques that we had not observed in our student analysts. Perhaps this is not surprising given that our students were typically not domain experts. Hence, we found the library analyst first attempting to judge the client's level of expertise in library automation, and then tutoring the client when necessary. We

saw a fair amount of client-convincing by the analyst. Finally we saw discussion of the environment where the system would reside, including the support or non-support of teachers, principals, administrators, and existing library staff. Along with this came discussions of cost and actual installation into the existing system.

2.4. Findings from Protocol 2

The nature of this protocol was different than 1 and 3. Here, we were interested in looking at how a library analyst would critique a textual description of a proposed system. The description was taken from a well-studied example in software specification circles, that of a library database system [6]. To give the reader a sense of the size of the problem and the informality of the text, the description called for a small library database system; the entire description fit on half a page.

The protocol setup was for the analyst to read the description, and critique it aloud. One of our group acted as an interrogator to ensure that all parts of the problem were discussed and to clear up ambiguities if and when encountered.

Because of the rather artificial nature of this protocol, little can be said about the general control model employed in critiquing. However, we did gain two valuable insights from the session. First, we began to see the type of deep domain model we would need if we wanted to produce the same type of critiques by machine[5] (see **A10, A15, A16** and **A17** in particular).

Second, and more germane to our other protocols, we saw frequent use of examples to back up a critique. Thus, we might see something like, "Gee, there are going to be problems with that. For instance, what happens when ..." (**A10, A15, A16**).

2.5. Findings from Protocol 3

General dialog control. The interview structure here was much less discernible than with protocol 1. We might roughly divide this session into an information gathering phase, a refinement phase, and finally a discussion of automation as a solution. However, each of these three types of behavior could be found intermixed within the session.

Acquisition and analysis techniques. While there is much behavior that we have yet to explore in detail in this protocol, two particular techniques used by the analyst occurred

[5]A fuller discussion of this topic, along with a detailed example, can be found in [3].

often enough to warrant an immediate look. The first is the use of examples. As discussed in section 2.1, our initial hypothesis was that examples are useful in gaining a more complete picture of the proposed system and of the environment in which it will reside. In fact, we did see this use of examples in this protocol. However, we also saw a variety of other uses including support for arguments, presentation of alternative solutions, demonstration of problems with the old (or proposed) system, showing advantages of the proposed system, and confirmation that the analyst understands some aspect of the problem. We have used this list as the beginning of a classification of the uses of example generation in problem acquisition (see discussion in Appendix A).

The second analysis technique that we found of interest in protocol 3 was that of summarization. We had no a priori hypothesis that summarization was a valuable problem acquisition technique. However, a study of the protocol convinced us that it was a powerful software analyst tool. We saw at least the first two types of summaries listed below being employed spontaneously by the software analyst; the third was brought about under our prompting.

1. Local summaries. Usually directly follows some statement of fact. Could sometimes be viewed as a paraphrase (**A25, A28**).

2. Topic summaries. When *enough* of a topic is filled in, a summary is produced. We do not currently have a good definition of *enough* (**A26, A27**).

3. Session summary. Under our prompting, the analyst produced a summary of the entire problem as he understood it (**A29**).

As with example generation, summarization can be used for more than one purpose. While the primary use of summarization was as a device to confirm that the analyst and the client had a common understanding of the problem, other uses were observed:

- Verification with the client that all aspects of a topic have been explored. In practice, this is stated as a question, e.g., "Is it true that A, B, and C are the only things we have to worry about in this system?" (**A27**).

- A trigger *for the analyst* to raise other related issues. That is, we see the analyst saying something like "OK, I think I see now. Your problem is X. Oh, that reminds me, I should ask you about Y." (**A29**).

- A trigger *for the client* to raise related issues. This is similar to the above point, only the client is now the one reminded of something that needs to be discussed (**A25, A28**).

It seems clear to us that a good summarization theory will need to address both of the generation problems presented above: *what type* of summary should we employ to achieve a particular acquisition goal, and *when* should we employ it to gain maximum effect. While we continue to study the problem, we have no such theory in hand.

Support of hypotheses. This protocol both supported our belief that examples play a key role in acquisition and analysis, and extended our view of their use. The protocol also tended to support our belief that an experienced analyst would work in concert with a client in generating a problem statement. In particular, the analyst was an active critic of proposed solutions, both his and the clients'. The main differences between this and protocol 1 were a) the amount of time devoted to traditional domain analysis activities, i.e., finding the objects, operations and constraints of the domain, and b) the general style of acquisition, here a little less disciplined and a little more opportunistic than in 1. However, we would not feel comfortable saying much more than this in comparison of the two without a more controlled experiment.

2.6. Reasons to disbelieve our findings

There are several factors that make our results suspect. First, we told both analysts and clients in protocols 1 and 3 that we would only be holding a one hour session with no follow up. It is not hard to believe that this affected the way the analysts structured their time. When asked after one of the sessions in protocol 1 what he would normally do now, the analyst replied he would do a site visit, talk to the staff, teachers, and principal, and generally do more information gathering. In summary, we feel we will need a more realistic protocol experiment to support a more refined view of software analysis.

Second, while our goal was to remain as passive observers in protocols 1 and 3, there were times during a session when we felt forced to answer a question or ask a question ourselves. The extent that such interference affected the behavior of analyst or client remains a question.

Finally, the sample in each protocol is small. Can we really generalize about the behavior of domain experts looking at only one such expert and two sessions? Further, is the problem of library automation characteristic of a large set of problems? We'd answer no to the first question -- individual differences must be taken into account -- and yes to the second -- resource management problems are ubiquitous. Thus, more compelling arguments must rest on further observations and experimentation in the domain.

2.7. Reasons to believe our findings

Many of the techniques seen in our protocols were seen across protocols, and hence analysts. In particular, each analyst used example generation and summarization at least to some degree. This gives us further confidence that it would be profitable to study these cross-domain cross-analyst techniques in more detail.

There is some reason to believe that the domain-expert analyst in protocol 1 was using a standard analysis technique. In particular, Boland [1] studied various system analyst styles used in MIS problems. Boland found that the traditional approach was one of *learn* about the client, *analyze* the data, and *make* suggestions. Boland argued that this approach was most often used when the analyst was viewed as the leader of the acquisition process. An alternative style suggested by Boland -- *teach* each other about their respective professional skills, *suggest solutions*, and *critique* each other's suggestions -- was seen as a sometimes better suited, cooperative, problem solving process. We would argue that the analyst in protocol 1 could be more closely viewed as using the traditional approach described by Boland (although traces of the alternative style can be seen).

Work by Soloway and Abelson [9] has some bearing here. Their study was of the design/implementation process that follows from an informal problem description, such as the one used in protocol 2. Their approach was to record think-out-loud protocols of a single designer as he or she began to implement a system. On the surface it would appear that their study was of the designers who use the type of specifications developed by our analysts, i.e., a study of implementation techniques as opposed to our study of acquisition and specification techniques. However, we find some interesting similarities:

- The Soloway&Abelson notion of mental simulation, i.e., one that supports a better understanding of the problem by the designer/analyst, was found most strongly in our transcript from protocol 3 (for instance, see **A18, A23, A29**)). This protocol can be viewed, in terms of the Soloway&Abelson study, as an analyst working on an unfamiliar problem in a familiar domain, and hence agrees with the findings of similar types of design problems studied in [9].

- The analyst in protocol 1 can be viewed as one working on a familiar problem in a familiar domain. As with the findings of similar design studies by Soloway&Abelson, little mental simulation *in support of the analyst's understanding of the problem* was seen in the two transcripts generated from protocol 1[6].

[6]This is not to say that "simulations" were not seen in this protocol. Indeed, they were frequent (see most of Protocol 1 in Appendix A). However, their main purpose appeared to be to convince the client of something. One view might be that they were an explicit attempt, by the analyst, to force the client to do a Soloway&Abelson type mental simulation to better understand the problem.

- Expert analysts and designers familiar with the problem and domain seem to jump quickly to a particular view of the problem, and then proceed with fine-tuning (see our discussion in 2.3 and III.C in [9]).

Of course, there were differences as well. However, these were not unexpected given the different nature of the two studies, i.e., specification versus implementation, think-out-loud versus client and analyst interaction.

Moving from general design to detailed implementation, the work by Kant [5] studies designers of low level, geometry type algorithms, again using think-out-loud protocols. As with the Soloway&Abelson study, we find similarities from what appear to be disparate development processes:

- Kant found both concrete execution and symbolic execution in her protocols. The analysis of our protocols supports this finding, but turns up more of a continuum between these two extremes, as can be seen from the examples in Appendix A.

- Kant also argued that concrete or symbolic execution of an algorithm may point to bugs, e.g., missing pieces, inconsistencies, that further drive the design process. Our use of the term summarization in this paper includes and supports this type of "difficulty driven" [5] design behavior. See **A29** for a good example of this from our protocols.

Finally, some distantly related work by Dietterich [2] involved studying the design techniques of expert Mechanical Engineers. His findings suggest that over a 10 hour design session, only the first half hour or so is spent exploring alternative solutions. The remainder of the time is spent patching what solution was chosen.

Of course, this raises an interesting question. Do experts spend so little time on exploring a space of alternatives because they can quickly trim down to a small and interesting set? Or is it simply that humans cannot compare and contrast a large number of alternatives in complex domains, and hence will always choose among a few standard approaches? In informal conversations with designers, they often say that they are happy with a *good* solution as opposed to the *best* solution (satisfy rather than optimize). If we give analysts and designers the right tools, can we expect that they can more often set their sights on the best solution?

In summary, if one views "design" as construction of a complex artifact, be it a specification, a high level implementation, an algorithm, or even a mechanical device, then one might hope that some general design techniques would cross application and process boundaries; such seems the case in the studies discussed here.

3. Conclusions

Our interest is in acquiring a problem description from a user, storing it in a form that allows us to reason about it, and doing our best to find problems with it before implementation starts This has lead us to work on a system that will contain the following components:

- **Component 1**: A model of the domain of interest.
- **Component 2**: A problem acquisition component that attempts to extract the details of a problem from a client.
- **Component 3**: A critic that attempts to poke holes in the current problem description.

This paper has focused on our efforts to understand the techniques used by experienced software analysts when acquiring a problem description from a client. We believe that this will eventually support Components 2 and 3 above.

We have described three protocols that were carried out in an effort to better understand the problem acquisition and software analysis process. We can summarize our findings from these protocols as follows[7]:

- Example generation is a key device in carrying out a variety of software analysis functions, including problem acquisition, problem debugging, and general argumentation and negotiation behavior (c.f. [8]).
- Summarization plays an important role not only in reaching a common understanding between analyst and client, but also as a trigger for new lines of acquisition and analysis.
- A major difference between an analyst that is a domain specialist and an analyst that is a generalist is the amount of time spent in acquisition versuses the amount of time spent a) exploring the social and environmental impact of a proposed solution, and b) convincing the user that it is a good solution.

In conclusion, we have gained a better understanding of what the interesting problems

[7]To reiterate, these findings must be tempered by our comments in section 2.6.

are in automating problem acquisition in KATE. We have also learned enough to begin to incorporate some of our findings into our demonstration system, giving us the ability to test them in a more controlled fashion. In particular, we are looking at the framework built by Rissland for constrained example generation [8] as a possible model for the example generation observed in our protocols.

We have confirmed some of our beliefs about the behavior of software analysts who are generalists, but have raised new questions about the behavior of domain specialists. In KATE, we have embraced the domain specialist point of view, and in particular, have begun to study how we might mechanize the type of environmental impact exploration seen in protocol 1, and the reasoned critique seen in protocol 2.

4. Acknowledgements

We would like to thank Sarah Douglas and Barbara Korando for their assistance in carrying out our protocol studies.

This work is supported by the National Science Foundation under grant DCR-8603893.

5. References

(1) Boland, R., Protocols of Interaction in the Design of Information Systems: An Evaluation of the Role of the Systems Analyst in Determining Information Requirements, Ph.D. Thesis, Case Western, 1976

(2) Dietterich, T., Ullman, D., Stauffer, L., Preliminary Results of an Experimental Study of the Mechanical Design Process, Technical Report 86-30-9, Computer Science Department, Oregon State University

(3) Fickas, S., Automating Analysis: An Example, In *Proceedings of the 4th International Workshop on Software Specification and Design*, Monterery, 1987

(4) Fickas, S., Automating the specification process, Technical Report 87-05, Computer Science Department, University of Oregon, November 1987

(5) Kant, E., Understanding and Automating Algorithm Design, *IEEE Transactions on Software Engineering*, November 1985

(6) Kemmerer, R., Testing formal specifications to detect design errors, *IEEE Transactions on Software Engineering*, January 1985

(7) London, P., Feather, M., Implementing specification freedoms, *Science of Computer Programming*, Number 2, 1982

(8) Rissland, E., Hypotheticals as Heuristic Device, In *Proceedings of AAAI-86*

(9) Soloway, E., Abelson, B., The Role of Domain Experience in Software Design, *IEEE Transactions on Software Engineering*, November 1985

APPENDIX A

There are two general problem acquisition techniques that are of particular interest to us, example generation and summarization. Below are representative examples of each taken from our 3 protocols. We have indexed each piece of dialog so that we may refer to it in the main body of the paper (see sections 2.3 - 2.5).

EXAMPLE GENERATION.

Our first attempts were to classify the different uses examples were put to during problem acquisition. From this, we derived categories such as *present alternatives, critique way things done now, show benefits of proposed system/solution, critique proposed system/solution,* and *lessen client's concerns.* We have also begun to look at the structure and content of the examples themselves. For instance, does an example use real or symbolic data? Does it rely on context or describe a complete scenario? Instead of choosing to organize around one or the other here (or employ some more elaborate index that gives both), we chose to simply list examples by protocol. To mitigate the appearance of a random selection, we will frequently include comments that set the use of the example within the context. We have also attempted to choose a collection that is representative of the various types of structure and content we observed.

Protocol 1

1. **Analyst**: "OK, how about on-line, public access, so that a student could come in and rather than go through the card catalog, could search your holdings on a terminal [*do you want this feature*]?"

2. **Analyst**: [*arguing for off the shelf software for a library automation system*] "Well, I would, I would strongly suggest that you not program, that your level of programming for that be limited to the utilization of the programming level of a database management system that exists. And the reason I say that is that the database management systems are well supported by either ASCII tape or microfilm or whoever puts the system out. And there are updates. And, uh, if you write, if you sit down at the programming level and you write this software that's going to be required here, the you're kind of dependent on the documentation of the programmers [*to understand the system*]. And those people being around, or sticking around to help you out. And I know, because I've also been the administrator of a system that's dependent on programmers passing that information on from generation to generation. And it doesn't work out well."

3. **Analyst**: "I was just thinking of different scenarios for your, for what you are looking at now. And the available technology. One of the, um, newest pieces of hardware in technology now that is applicable to libraries is CD ROM. And I think that in the kind of system that you are talking about here, it certainly has some possibilities. [*begin tutoring*] They have CD ROMs now that are write-once, which means you can write to them once and then use them many times [*end tutoring*] ... in terms of the union catalog, you could easily include the indices on a CD ROM. And CD ROM readers for microcomputers are not very expensive."

4. **Analyst**: [*pointing out problem with card catalog*] "Now, the only problem you have is everybody wants to come in and look under *International*, then they have to queue up and wait for the I-drawers, OK. [*goes on to discuss benefits of automating card catalog - see below*]"

5. **Analyst**: [*discussing benefits of on-line card catalog integrated with acquisitions*] "... so that as you checked in serials, say, and a patron was looking at the catalog, they could see that the last issue you checked in was November 13th issue of Newsweek."

6. **Analyst**: [*discussing problems with use of central database*] "You could put your union catalog on a mainframe, and everyone would have dialing capabilities to that. You would have to look at the costs involved there ... you'd have to see what kinds of loads you'd have there."

7. **Analyst**: [*answering client's concern about cost of conversion to automated system*] "City X, for instance, started out and converted their records as they were used ... As it works out, many of the schools in the System of Higher Education have their, are well under way in converting their entire collection ... You could probably handle your system with a microcomputer and circulation easily with a, with say, a single microcomputer workstation." [*This was not a particularly satisfying answer to us, but the client did not question it.*]

8. **Analyst**: [*exploring cost/benefit issues*] " If you wanted to have an integrated system ... not only know that this high school has this title, but know whether it's checked out or not ... or if they've got this serial title, what their latest issue or if it's checked in ... You are going to have to look at it and say, 'OK, what, how much of an investment is this resource sharing worth?'"

9. **Analyst**: [*giving scenario where problems might arise*] "And one of the problems you're going to have is that when you put a new record on, or when you use a record that's already there, if you don't duplicate all the information that you had before on, all the local information, then depending on how your system of merging is set up, that information could be lost."

Protocol 2

10. **Analyst**: "Since we are in a campus setting, talking about a departmental library, I would have to ask what ordinary borrowers are we distinguishing

between ... For instance, a senior working on a honors thesis may need the same type of borrowing privileges that a graduate student needs [*do you agree?*]."

11. **Analyst:** [*confirming meaning of "removal" of book*] "OK, and then removing it would be, say, it gets old, or it gets torn up or it's missing pages [*client agrees with meaning*]."

12. **Analyst:** "[*arguing against including a query in an automated library system that allows one to see who last checked out a book*] It used to be in most libraries that most people signed their name on a card. And you could look up and down the card and it was kind of interesting to see for your years [*of attendance*], and also, gaps of time, like 10 years may have elapsed and then there was a big use of it again.
... and it was kind of fun to see that your old professor read this book 20 years ago or something and had signed their name to it. Those days are gone, and I think it's partly because the value was not there in terms of knowing who had the book last. What point is it, once the book is returned, who needs to know who had it last, and why [*do they need to know*]?"

13. **Analyst:** [*discussing how to handle high demand books*] "Well, there are several little things that are possible. One of them is that in a departmental library where a book needs to be replaced frequently, uh, chances are very good that the book is going to go out of print and you might want to buy 2 or 3 copies or you might want to limit circulation. You might also want to look at keeping it, in some cases, in a place where it won't get stolen and then check it out like you do a reserve item."

14. **Analyst:** "There's no departmental library in any university in the country that I know of that has the budget to acquire all the books in their discipline. And so having access to, say, the on-line database for the Center for Research Libraries or to other records of straight books, not even journal articles, but it seems to me that that would help expand your holdings."

15. **Analyst:** [*pointing out problems with honor system for check out*] "Generally, check in and check out systems don't operate very well on the honor system. Not because people are not honorable, but because it means stopping and doing something in a detail that has nothing to do with your research needs. And often people forget, or they say I'm just going back to my office for a couple of minutes and I'll bring it right back, or "I'll bring it back tomorrow morning and it will still be on the shelf.""

16. **Analyst:** [*pointing security problems in an automated library system*] "Occasionally, someone else comes in and says, 'I want a list of all the books so-and-so has checked out.' That becomes a little more difficult in the ethical considerations of it ... A good example is ... [*goes on to describe animal rights group that liberated lab animals on campus*] Somebody wanted to know who'd been checking out books on animal rights."

17. **Analyst:** [*pointing out intimidation problems given free access to information*] "Often in a larger setting faculty say, 'Nobody, none of us have secrets from each other and we should all be able to know who has what, etc.' On the other hand, there have been graduate students who say they don't like getting called Saturday morning at 8 am saying 'Bring in this book.' by a person who could

affect their future."

Protocol 3

18. **Analyst**: "So, upon receiving the application, then you would start entering this information as it comes in. And that would also be able to automatically generate letters of missing items at certain dates."

19. **Analyst**: [*discussing how easy system would be to use*] "... what if the option was you had, you could just type in somebody's name and all their information would come on the screen at that point."

20. **Analyst**: "See, one advantage about the automation of that is that it's real easy to get out lists of, I mean you could have it type out a list of who was admitted Fall quarter and so forth ... you can access the information by person's name, alphabetical or by term ... It would be real easy to generate other sorts of lists, admitted, rejected ..."

21. **Analyst**: [*noting problems with putting school transcripts online*] "It seems like it would be excessive to actually enter the entire transcript for people [*who have gone to 6 schools*]".

22. **Analyst**: [*in answer to a concern about duplication of information*] "It seems like you could do it just one place. Just enter it on the computer. You could even have the computer generate the card file; it could type out those cards."

23. **Analyst**: "And if that's on the computer, then it would be real easy to generate other sorts of lists ... of admitted, rejected, all that ... letters and so forth, correspondence."

24. **Analyst**: [*answering client's concern about data entry*] "I mean, you could have work study students entering the information ..."

SUMMARIZATION.

Summarization is seen as playing a multi-purpose role in the protocols. We are now looking at summarization styles (list elements in a set, describe a process, paraphrase) and frequencies (after every acquisition of new information, at the end of topics). We will provide a few illustrative examples below, and at the same time note that we have not progressed far enough to give a useful classification of types.

25. **Analyst**: "How many, what's your patron count? How many people use the library?"

Client: "1300."

Analyst: "About 1300 in the school. OK." [*Our supposition here is that the analyst paraphrased the response because it was a surprising answer taken literally, i.e., that 1300 students of a high school were library users. More likely the client was giving the total number of potential users.*]

Client: "We have about, I'd say, about 250 check-outs a week." [*The client now volunteers more detailed information.*]

Analyst: "OK. And you average about ..."

Client: "40 a day."

Analyst: "About 40 a day."

Client: "Yes."

Analyst: "so about 200 a week." [*Further refinement of the 250 estimate above.*]

26. **Client**: "Everything is centralized."

Analyst: "OK. So they order things for you ..."

Client: "Um-hmm."

Analyst: "and then process them ..."

Client: "Yes."

Analyst: "and so you get a book that's been processed and has got a book card in it ..."

Client: "Um-hmm."

Analyst: "a check-out card and it has the set of cards and you file those."

Client: "Right."

27. **Analyst**: "And so a student's file isn't considered complete until they have all the things. Are there other things other than letters of recommendation, test scores, statement of goals, transcript, and application?"

Client: "No."

28. **Analyst**: "So the log file has information about who were, which students were admitted which term, and so forth."

Client: "It [*the file*] is divided into terms."

29. **Analyst**: "OK. Um. I don't know how far you want us to take this, at this point. I think I know basically what the process of graduate students admissions and application is." [*This said 40 minutes into the session.*]

Observer: "You want to summarize it?"

Analyst: "Yeah. Um. Basically the process involves completing a file for each graduate student. That file contains letters of recommendation, test scores, statement of goals, transcripts, and the application itself. Um, and in, while that information is coming in you keep track in the file folder itself what parts of that information are missing and have yet to come in. And basically there is

also a card file which is an alphabetical list that you use to help locate each graduate student's file folder. And this file . . . um, I'll talk about that later. And also there's a log file that's by term, and it's information for each person that's applied for the term ... [*continues summary*] ... and then you send the file to . . . well, and you also tell them when it's going to appear before GAC. So, is that the same time each term or how does that . . .?"

Client: "Within a few weeks, yeah."

Analyst: " ... " [*Thus begins a whole new round of discussion lasting another 40 mintues. This "new" session is generally focused on filling in details left unexplored in the previous interaction with the client.*]