Iris Vessey and Sue A. Conger

Requirements Specification:

Learning Object, Process, and Data Methodologies

Specifying information requirements—determining and documenting the requirements for an information system, is arguably the key to developing successful information systems (IS). Since this is the first step in the systems development process, it is clear that not doing it effectively will have significant impact on both the effectiveness and the efficiency of the resultant system. Not getting the correct final system requirements initially is largely responsible for the cost and schedule overruns that are still fairly prevalent in IS development.

Specifying information requirements is not only the most important step in developing IS, it is probably also the most difficult. The crucial aspect of the process is to develop a mental model of the system (i.e., to determine *what* the system needs to do) [6]. Once the information requirements of the new IS have been specified, the development process consists of successive transformations of that initial model until, finally, a computer-based solution to the problem is achieved.

Specifying requirements is a complex task that is difficult for systems analysts to address, due to human problem-solving limitations [5]. Short-term memory is limited in capacity; transfer from short-term to long-term memory requires a relatively long time, approximately 5 to 10 seconds; storage and retrieval mechanisms are fallible. Short-term memory capacity is perhaps the most devastating aspect of these limitations. Classical estimates of short-term memory capacity suggested that problem solvers can effectively handle 7 ± 2 chunks of information [14], but this is an upper limit that may be drastically reduced by interference. To overcome these limitations, human problem solvers resort to various ways of effectively increasing the capacity of short-term memory as well as using external memory devices and/or techniques to support working memory.

The aid most commonly used to support working memory in information requirements specification is a systems development methodology and the associated representation technique. By methodology, we mean a systematic approach to the task of systems development (see [16]). Early methodologies provided guidelines mostly in the form of a set of standardized activities and standardized forms to be completed; little effort was devoted to handling complexity explicitly. Subsequently, the major thrust of methodologies has been the desire to address complexity directly [6, 8, 10].

Despite the significance of information requirements specification to the successful design of systems and the large body of descriptive and conceptual literature on methodologies, there have been surprisingly few empirical studies conducted on the methodologies themselves. In fact, review of the literature revealed only one such study, by Yadav et al. [24], who investigated the effectiveness of data flow diagramming [8] and a variant of the systems analysis and design technique (SADT) [20] in specifying information requirements. They report that although data flow diagrams are easier to learn and to use, neither produced significantly better specifications.

Given the importance of methodologies to IS development and the dearth of empirical evidence regarding their use, we conducted an exploratory study to investigate the performance of three methodologies in aiding novice systems analysts who were learning to specify information requirements. The methodologies investigated were the structured techniques [6, 8, 25], Jackson System Development (JSD) [10], and the object-oriented approach [3]. We investigated the structured techniques and JSD because they are currently the most widely used structured analysis methodologies (see Necco et al.) [15]. Since object-oriented approaches to systems development are receiving considerable attention in the software engineering arena and since some firms, such as Arthur Andersen Consulting, are moving toward replacing their current methods with object-oriented methods, it seemed appropriate to investigate the relative merits of an object-oriented approach in a business setting.

Methodologies

Humans are limited information processors. Specifying information requirements is extremely difficult for human problem solvers because it requires handling large amounts of

knowledge. Such problems are usually handled by decomposing the area under investigation into subproblems for which solutions can be found or generated [21]. The key to successful problem decomposition lies in structuring the problem in such a way that the subproblems can be integrated to form the solution for the complete problem.

In the systems development domain, decomposition is usually achieved by applying methodologies that are designed to formally address ill-structured systems development tasks. There are many choices for decomposition, since many different types of information must be propagated through a system. In practice, the types of decomposition employed in methodologies are relatively few in number and employ two bases for decomposition: process and data. The structured techniques of DeMarco [6], Gane and Sarson [8], and Yourdon [25], for example, employ process decomposition, which views a system from an input-process-output perspective. JSD [10] is more data oriented. Currently, methodologies are being developed that employ object-oriented decomposition [3, 4].

Structured analysis is based on the concept of top-down partitioning or decomposition of systems based on "processes." Requirements are specified using data flow diagrams, a data dictionary, and process specifications [6, 8]. Data flow diagramming is the technique used to represent the hierarchical decomposition of the real-world system under investigation. Specifically, data flow diagramming achieves top-down partitioning by decomposing the system first into subsystems, then into processes performed within a subsystem; each of those processes is ultimately specified as a processing cycle. For example, a manufacturing enterprise would have an order subsystem to track orders for the goods it manufactures; the order subsystem would consist of a number of processes, such as create order, cancel order, amend order. Each of these functions goes through a series of processing steps (for example, creating an order might involve validating order details, checking customer credit status, updating the order file, and so on). Hence, the structured techniques emphasize the processes that transform the data. Data flows are shown as inputs and/or outputs to the subsystems, processes, or steps in the processing cycle. The data dictionary contains definitions of the data in the system (flows and stores). Process specifications are developed for the lowest level or primitive processes on the set of data flow diagrams.

JSD, despite being known as a data-oriented approach to systems development, in effect emphasizes both data and processes, and is conceptually similar to the object-oriented approach, In JSD, requirements are specified using 1) an entity-action list, which identifies the entities in the systems and the actions that are performed on them and the actions that they themselves perform, 2) an entity-structure diagram, which shows the order of actions for each entity, and 3) an initial model (Jackson System Diagram), which connects real-world processes to model processes. Additionally, to provide specifications comparable across methodologies, the novice analysts in this study were required to produce a data dictionary, which does not form part of Jackson's methodology. Further, rather than annotating the process boxes in the initial model with sys-0, sys-1, and so forth, which have no significance to the problem at hand, the novice analysts were instructed to annotate the model with the actions from the entity-structure diagram.

Object-oriented analysis and design uses the concept of an object as the unit of decomposition [3]. An object is a component of the real world that is mapped into the software domain. It is an entity that is characterized by the actions that are imposed on it and the actions it imposes on other objects. Hence, an object contains both data structures and the allowable set of operations on the data. The process of object-oriented design interleaves analysis and design of objects with analysis and design of operations relating to objects. The object-oriented approach to systems development, then, provides a more balanced treatment of the objects and operations that exist in the model of the real world than the process approach. Further, object-oriented development encompasses explicit definition of message input and message output for each object.

In the object-oriented methodology used in this research[1] [3], requirements are specified using a series of lists. These are the object list, action list, object-attribute list, and action-attribute list. The final system design is represented as a "Booch" diagram. Object-oriented analysis as described by Booch uses Ada constructs to define messages between objects. In doing so, several conceptual leaps are taken in the design that require making changes in the previously defined tables (i.e., object lists, action lists, and sometimes the attribute lists). The instruction in object-oriented analysis for this experiment had the novice analysts construct a further table, known as the message list, that identified 1) the calling object, 2) the called object/action, 3) the input to the called object, 4) the output from the called object/action, and 5) whether the called module does read-only or write processing.[2]

The Study

We investigated the effectiveness of the process-, data-, and object-oriented methodologies in specifying information requirements. We used process-tracing techniques, which provide data about what happens during a problem-solving exercise, rather than simply giving participants a stimulus and measuring the outcome once problem solving was complete. Input/output analysis permits assessing only problem-solving outcomes, such as time taken to develop a solution to the problem and the quality of that solution. Process-tracing techniques, on the other hand, also permitted us to determine certain characteristics of the problem-solving processes used. Hence, pro-

[1]Note that, although Booch used the term "object-oriented design" rather than "object-oriented analysis and design," it is clear that he regarded his methodology as incorporating analysis as well as design, since he positioned it as a mapping from the problem domain to the solution domain.

[2]Note that neither the changes made to JSD nor to Booch's object-oriented methodology affected the integrity of the methodologies. The changes were made simply to aid novice analysts to apply the methodologies.

cess-tracing data is far richer than outcome data.

Our study examined the performance of novice systems analysts. There are several reasons. First, the major objective of this study was to examine the effect of methodologies on learning to specify information requirements. In this context, novice systems analysts have not previously been exposed to any formal systems development methods (i.e., they do not possess a large store of alternative methods). Hence, it is easier to teach them to apply a specific methodology than it is to teach new methods to people who may already be experts in developing systems, either by their own efforts or by using a given methodology. Further, experienced problem solvers might prefer to use the methods they use in practice rather than the given methodology. Second, examining expert problem solving can be quite difficult, since experts automate their problem-solving processes to the point at which they are no longer able to articulate what they are doing [2].[3] Novice problem solving, on the other hand, is more amenable to investigation.

Further, this study investigated learning over time. It is important not only to assess the performance of methodologies immediately following instruction but also to determine whether certain methodologies differentially facilitate the ability to learn them. This study investigated the following research questions:

- Are novice systems analysts able to develop information requirements specifications more readily with certain methodologies than with others?
- Do novice analysts learn to use certain methodologies more readily than others?

Method

To explore the preceding research questions, a process-tracing study was conducted in which novice systems analysts specified the information requirements for each of three applications. Protocol analysis was the process-tracing technique used, since it provides richer insights into processes than alternative process-tracing approaches (see Ericsson and Simon [7] for a discussion of protocol analysis and the validity of protocol data). The problem-solving sessions were videotaped, providing a visual as well as a verbal record of problem-solving behavior.

Participants developed information requirements specifications using a given methodology (i.e., we used a repeated-measures design). Participants were randomly assigned to a methodology. The order of presentation of two of the three applications was counterbalanced. The remaining application, which was not analyzed in this research, was always completed second. The data from the first and third trials were analyzed to better differentiate the participants' attempts at learning to use a methodology.

Two applications were used as stimulus materials in the experiment. The applications were chosen to be of roughly equivalent complexity and to be completed within two hours.[4] Standard sets of training materials were developed for each methodology to ensure comparable levels of training. Participants were provided with materials on 1) what the methodology entailed, which included the steps involved in the methodology, the information recorded at each step, and the format in which it is recorded. We refer to this material as declarative knowledge, i.e., it is knowledge about *what* needs to be accomplished at each step to fulfill the requirements of the methodology; and 2) how to apply the methodology (this material was presented in the form of the solution to a problem using the given methodology). By examining the example solution, participants could see *how* the methodology was applied in a given instance. We refer to this material as procedural knowledge. The volume of materials on methodologies presented to participants was similar for each methodology with approximately 5 to 6 pages of material on each of the declarative and procedural aspects.

Of necessity, the number of participants in protocol analysis studies is small. Participants were six students enrolled in a course on software engineering in the business school of a large university. Participants received instruction in a number of methodologies as part of their course work. The students had no prior experience in using any of the methodologies investigated in the study. Methodology knowledge at the time of the study was assessed via a background questionnaire. On a scale of 1 ("not very familiar") to 7 ("extremely familiar"), participants using the process, data, and object methodologies reported their level of knowledge of the methodology they used in the study as 4, 4, and 3.4, respectively.

The novice analysts received in-class instruction, completed in-class exercises, and a homework exercise in each of the methodologies.

At the experimental session, participants reviewed both the standard instructional materials (the declarative aspect of the methodology) and the application presented using the given methodology (the procedural aspect of the methodology) prior to the problem-solving session. They were then videotaped as they specified the information requirements for the three applications using the given methodology. During the study session, they could refer to the class materials on methodology as well as the example solution. Participants were permitted breaks following the completion of each requirements specification. Finally, they were debriefed regarding their performance, the difficulties they had encountered, the ways in which they overcame those difficulties, and their thoughts on the experimental session. The sessions, with requirements specification for three applications and two breaks, lasted from 3.75 to 6.35 hours.

Two types of formal analyses were conducted on the protocol data. First, the protocol data was coded to reflect the time spent referencing certain documents. The documents examined in the study were 1) the problem statement (i.e., the application), 2) the notes on the steps that comprise the methodology (declarative methodol-

[3]Intuitively, consider the difficulty of trying to communicate to a nondriver how to drive an automobile. Once we have learned to drive, we lose the ability to describe exactly how we do it while at the same time keeping abreast of traffic conditions, carrying on a conversation with those travelling with us, and so on.

[4]The applications are available from the authors upon request.

ogy knowledge), 3) the example solution specified using the given methodology (procedural methodology knowledge), and 4) the resultant specification or solution. Since the novice systems analysts who participated in this study were in the process of learning the methodologies they had been taught, we expected they would still need to acquire both declarative and procedural knowledge related to those methodologies. Hence, in our analyses, we inferred that the novice analysts applied the knowledge acquired.

Second, the protocols were examined for the breakdowns that occurred during problem solving. Breakdowns are difficulties that occur during the problem-solving process [9]. In this instance, they can be viewed as impediments or obstacles to successfully specifying requirements.[5] Breakdowns were characterized as due to problems in 1) formulating a mental model of the application, 2) carrying out the steps of the methodology, and 3) applying the methodology to a particular application to reach a solution. Further, a breakdown in any of these categories may be due to one of three factors: lack of knowledge, incomplete knowledge, or incorrect knowledge.

[5]Note that breakdowns are an indication of the effectiveness of the problem-solving process. Since unresolved breakdowns will result in problems in the final requirements specifications, they can be used to infer the quality of those specifications.

Table 1 presents the types of breakdowns that may occur in each of the categories. Breakdowns resulted from lack of ability to formulate a mental model in one of the above categories, or the formulation of an incomplete or incorrect mental model. For example, lack of ability to formulate a mental model of the declarative knowledge relating to the methodology means that the analyst does not know what steps to perform; having incomplete declarative methodology knowledge means the analyst misses a step in applying the methodology. Formulating an incorrect mental model of the declarative methodology means the analyst performs a step of the methodology out of sequence, or performs a step that is not part of the methodology, hence misinterpreting what needs to be done next in applying the methodology. Similarly, lack of ability to formulate a mental model of the procedural knowledge relating to the methodology means the analyst does not know how to apply the methodology to specify information requirements. Having incomplete procedural methodology knowledge means the analyst does not complete a particular step in the methodology. Formulating an incorrect mental model of the procedural methodology means the analyst performs a step of the methodology incorrectly.

The breakdown results are presented as made up of three components: 1) the number of breakdowns the participant actually committed, 2) the number of breakdowns from which the participant recovered, and 3) the number of breakdowns from which the participant did not recover.[6] A recovery occurred when a participant corrected, or in some way overcame, a breakdown committed earlier in that part of the problem-solving session.

Since the number of participants in protocol analysis studies is always small, we used multiple outcome measures, as well as a repeated-measures design. If the multiple measures suggested essentially similar effects, we could feel reasonably confident we had indeed identified an underlying phenomenon.

Reliability analyses were conducted on the breakdown and the timing data. Intercoder reliability assessment was conducted on the breakdown data, since analyzing problem-solving breakdowns is, to a certain extent, subjective. Test-retest reliability was conducted on the timing data. Since coding visual observations from video tapes is largely objective, it was not considered particularly fruitful to have a second coder also code the data. The same coder, therefore, recoded certain protocols six months after the initial coding.

[6]The last figure is presented for simplicity. It can be obtained by subtracting the number of breakdowns from which the participant recovered from the total number of breakdowns the participant committed (i.e., the third figure equals the first minus the second).

Table 1. Problem-solving breakdowns

Mental model	Application	Declarative methodology	Procedural methodology
Lack	unable to form a mental model of the application	does not know what steps to perform	cannot apply the method to obtain a representation of the solution
Incomplete	mental model does not address the entire problem	skips a step	does not complete a step
Incorrect	mental model does not match the problem	performs a step out of sequence performs a step that is not part of the methodology misinterprets what is to be done	does the step incorrectly incorrectly integrates the current step with previous steps

Table 2. Reliability assessment of breakdown data

Protocol	Type of breakdown	Number of breakdowns	
		Coder 1	Coder 2
S4 (trial 3)	Application	2/0/2[1]	2/0/2
	Declarative methodology	2/0/2	1/0/1
	Procedural methodology	1/0/1	1/0/1
S6 (trial 3)	Application	0/0/0	0/0/0
	Declarative methodology	0/0/0	0/0/0
	Procedural methodology	3/1/2	3/1/2
S12 (trial 1)	Application	0/0/0	0/0/0
	Declarative methodology	4/0/4	5/1/4
	Procedural methodology	4/0/4	4/0/4

[1]*The entries in each column are:*
1. the number of breakdowns
2. the number of breakdowns subjects resolved
3. the number of unresolved breakdowns

Reliability was assessed for one protocol selected randomly from each of the methodologies. The same protocols were used for the breakdown and timing reliability assessments. Tables 2 and 3 present the results. Scanning of Tables 2 and 3 shows that both the breakdown and timing data are reliable. The data derived from the first coding were used, in both instances, in the ensuing analyses. Tables 4 and 5 present the results of the breakdown and timing analyses, respectively.

Results

We first assessed research question 1: whether novice analysts are able to specify information requirements more readily with certain of the methodologies investigated. To do this, we examined the results of the breakdown and timing analyses for the first trial. Results show that participants found the object methodology difficult, and the process and data methodologies much easier to apply.

We used Anderson's learning theory [2] as the basis for investigating the ability of novice systems analysts to use the three methodologies. This theory suggests that declarative knowledge is acquired first, and that procedural knowledge can be assimilated only following the assimilation of declarative knowledge. Hence, we examined the extent to which novice analysts made declarative and procedural breakdowns and sought declarative and procedural knowledge in specifying information requirements.

Breakdown analysis for trial 1 provides substantial evidence of the difficulty in using the object methodology compared with the process and data methodologies. Although the object methodology resulted in similar numbers of breakdowns overall (17) as the data methodology (18), object participants were less able to resolve breakdowns (0 compared with 9), leaving 17 and 9 unresolved breakdowns for the object and data methodologies, respectively. Participants using the process methodology com-

Table 3. Reliability assessment of timing data

Protocol	Document	Percentage time	
		Time 1	Time 2
S4 (trial 3)	Application	33.1	32.9
	Declarative methodology	0.0	0.0
	Procedural methodology	.2	.2
	Solution	66.7	66.8
S6 (trial 3)	Application	34.5	33.7
	Declarative methodology	4.2	4.0
	Procedural methodology	.5	.4
	Solution	60.8	61.9
S12 (trial 1)	Application	27.6	28.0
	Declarative methodology	16.8	16.4
	Procedural methodology	0.0	0.0
	Solution	55.6	55.6

mitted fewer breakdowns overall (12) than the data and object methodologies (18 and 17, as stated previously); they resolved 4 of the breakdowns, leaving 8 unresolved. These results suggest that the novice analysts investigated had much greater difficulty applying the object methodology than the data and process methodologies.

This finding is substantiated by examining the different types of breakdowns that occurred for each type of methodology knowledge (i.e., declarative and procedural). Results show that the object methodology was associated with substantially more initial and unresolved breakdowns in each of the declarative and procedural methodology categories than the other two methodologies. The numbers of unresolved declarative and procedural methodology breakdowns are shown graphically in Figure 1a. The data methodology resulted in low numbers of unresolved declarative methodology breakdowns compared with the process and object methodologies (1 compared with 4 and 6). These figures suggest the novice analysts had a better grasp of the declarative aspects of the data methodology than of the process and object methodologies. Note, however, the low number of procedural breakdowns for the process methodology compared with both the data and object methodologies (4 compared with 8 and 10). These figures suggest the novice analysts applied the process methodology much more effectively than the data and object methodologies.

From the viewpoint of the timing analyses, it is interesting to note the inverse relationship between the time spent referencing declarative and procedural methodology knowledge. See Figure 1b. The time spent on declarative methodology knowledge was 0.0, 3.7, and 10.1 minutes for the process, data, and object methodologies, respectively, while that spent on procedural methodology knowledge was 7.3, 1.5, and 0.0 minutes, respectively. Note that novice analysts using the object methodology did not examine the example solution at all, suggesting perhaps, that they did not understand the methodology itself, let alone how to apply it. Based on the notion that novice analysts who spend more time referencing material on the procedural rather than the declarative aspects of a methodology are able to apply the methodology more effectively [2], the results suggest that novice analysts are better able to apply the process methodology and least able to apply the object methodology.

Hence, in summarizing evidence relating to research question 1, we see that the breakdown and timing analyses both provide convincing evidence

Table 4. Breakdowns and recoveries in specifying information requirements

	Totals			Performance over Time					
				Trial 1			Trial 3		
	Appln[1]	DecMeth	PrMeth	Appln	DecMeth	PrMeth	Appln	DecMeth	PrMeth
Process									
S4	5/3/2[2]	7/1/6	5/0/5	3/3/0	5/1/4	4/0/4	2/0/2	2/0/2	1/0/1
S11	0/0/0	0/0/0	1/0/1	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	1/1/0
	5/3/2	7/1/6 (18/5/13)	6/1/5	3/3/0	5/1/4 (12/4/8)	4/0/4	2/0/2	2/0/2 (6/1/5)	2/1/1
Data									
S6	2/2/0	6/6/0	5/2/3	2/2/0	6/6/0	2/1/1	0/0/0	0/0/0	3/1/2
S10	0/0/0	2/0/2	12/0/12	0/0/0	1/0/1	7/0/7	0/0/0	1/0/1	5/0/5
	2/2/0	8/6/2 (27/10/17)	17/2/15	2/2/0	7/6/1 (18/9/9)	9/1/8	0/0/0	1/0/1 (9/1/8)	8/1/7
Object									
S3	6/2/4	4/0/4	11/1/10	1/0/1	2/0/2	6/0/6	5/2/3	2/0/2	5/1/4
S12	0/0/0	7/0/7	10/0/10	0/0/0	4/0/4	4/0/4	0/0/0	3/0/3	6/0/6
	6/2/4	11/0/11 (38/3/35)	21/1/20	1/0/1	6/0/6 (17/0/17)	10/0/10	5/2/3	5/0/5 (21/3/18)	11/1/10
Overall									
	13/7/6	26/7/19 (83/18/65)	44/4/40	6/5/1	18/7/11 (47/13/34)	23/1/22	7/2/5	8/0/8 (36/5/31)	21/3/18

[1] *The headings are the types of breakdowns committed:*
Appln = application
DecMeth = declarative methodology knowledge
PrMeth = example solution using given methodology (procedural methodology knowledge)

[2] *The entries in each column are:*
1. the number of breakdowns
2. the number of breakdowns subjects resolved
3. the number of unresolved breakdowns
Hence, (3) = (1) − (2)

The numbers in brackets represent the totals under that heading.

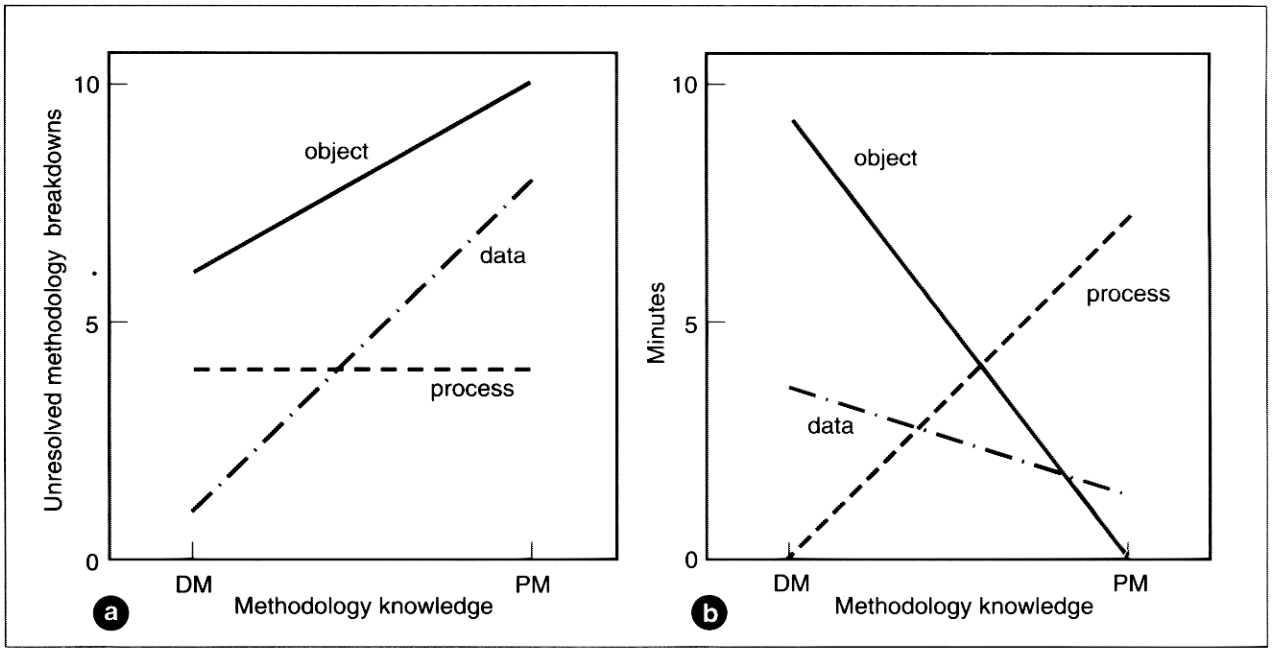

Figure 1. a. Number of unresolved declarative and procedural methodology breakdowns on trial 1; b. Time spent referencing declarative and procedural methodology knowledge on trial 1

that the object methodology is the most difficult of the methodologies examined for novice systems analysts to apply. Further, novice analysts found the process methodology easier to apply than the data methodology.

We next assessed research question 2: whether certain methodologies are easier for novice systems analysts to learn than others. Requiring participants to specify the information requirements for each of three applications permitted us to examine how the ability of the novice analysts to use the methodologies changed over time (in this case, from trial 1 to trial 3). Results suggest the process methodology was the easiest for novice analysts to learn over time, followed by the data, and then the object methodology.

The first piece of evidence on ease of learning comes from training times. To train novice analysts to the same level in the process, data, and object methodologies required 5, 7, and 10 hours, respectively. Hence, the object method required a training session twice as long as the process approach, while that for the data methodology was intermediate between the two.

Second, we can assess whether there are differences in learning to use the methodologies by examining the number of methodology breakdowns that occurred over time (i.e., from trial 1 to trial 3). Figure 2 presents both committed and unresolved methodology breakdowns for trials 1 and 3. The process and data methodologies experienced fewer committed methodology breakdowns over time (9 to 4 and 16 to 9) compared with the object methodology (16 to 16), which was static. The only difference in this pattern of results for unresolved breakdowns was that the results for the data methodology were relatively static over time (9 to 8), resulting in a pattern more similar to that for the object methodology than for the process methodology. These results suggest the process methodology is not only easier for novice analysts to learn initially, but also that it is the only methodology with which they improve over time (i.e., for which learning occurs), at least in the early stages of learning.

Further in-depth analyses were conducted by examining the number of declarative and procedural breakdowns that occurred over time for each methodology (see Figure 3). With the process methodology, unresolved declarative methodology breakdowns decreased from 4 to 2 and unresolved procedural methodology breakdowns from 4 to 1 from trial 1 to 3, indicating a marked improvement in ability to learn the methodology. With the data methodology, the number of unresolved declarative breakdowns was unchanged at 1 over time, while the number of unresolved procedural breakdowns decreased from 8 to 7. Hence, there was little consistent improvement in novice analysts' ability to learn the data methodology over time. Similarly, for the object methodology, the number of unresolved declarative methodology breakdowns decreased from 6 to 5 over the three trials, while the number of unresolved procedural methodology breakdowns was unchanged at 10. These results, therefore, indicate no overall improvement in learning the object methodology over three trials. In summary, note that, for the more critical procedural methodology breakdowns, there was a considerable decrease over time for the process methodology, a slight decrease for the data methodology, and no decrease for the object methodology.

Third, the timing analyses also suggest the process methodology was easiest for the participants to learn, followed by the data methodology. The mean times spent referencing the methodology on trials 1 and 3 were 7.3 and 0.5 minutes for the process methodology, 5.2 and 2.0 minutes for the data methodology, and

10.1 and 3.7 minutes for the object methodology. Hence, while all participants spent less time referencing the materials on methodology on the third trial, time spent on process, data, and object methodology knowledge decreased over 3 trials to 7%, 37%, and 38%, respectively. These figures show that greatest learning occurred for the process methodology, with much less (and similar) levels of learning occurring for the data and object methodologies.

Hence, a substantial amount of evidence suggests that novice analysts' learning progressed more rapidly with the process methodology. There was little consistent evidence of learning for either the data or the object methodologies. Although data and object participants needed to refer to supplementary materials on the methodology less frequently over three trials, they still committed, and were unable to resolve, similar numbers of the critical, procedural breakdowns. In response to research question 2, we see that the process methodology was the only one for which substantive learning occurred over time, with some learning occurring for the data methodology.

We can investigate further the power of procedural knowledge in problem solving. Looking at the numbers of declarative and procedural methodology breakdowns revealed that procedural breakdowns outnumbered declarative breakdowns by factors of 1.7 (44/26) for total breakdowns and 2.1 (40/19) for unresolved breakdowns. Hence, we investigated whether participants who acquired procedural methodology knowledge during the study were more effective in specifying information requirements than those who acquired declarative methodology knowledge.

We first categorized participants according to whether they sought predominantly declarative or procedural methodology knowledge to aid them during problem solving. Examining the verbal protocols reveals that subjects S1, S4, and S6 referred exclusively to the materials on the steps involved in the methodology (declarative methodology knowledge—a total of 34.9 minutes), while S2, S3, and S5 referred primarily to the example solution (procedural methodology knowledge—17.8 minutes; declarative methodology knowledge—4.7 minutes). We see that participants who used procedural knowledge quite extensively had a total of 24/10/14 methodology breakdowns.

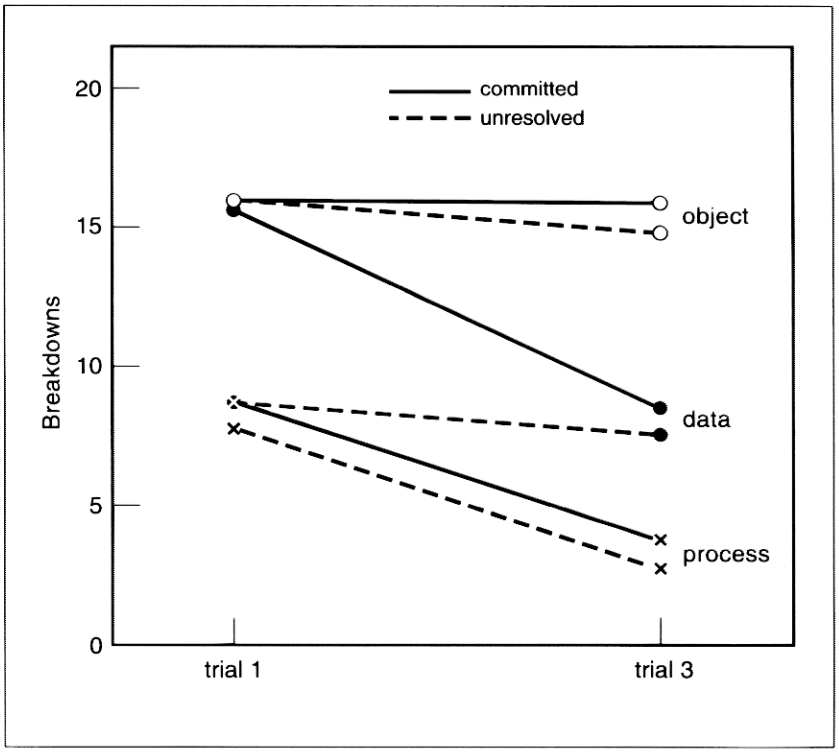

Figure 2. Committed and unresolved methodology breakdowns for each methodology from trial 1 to trial 3

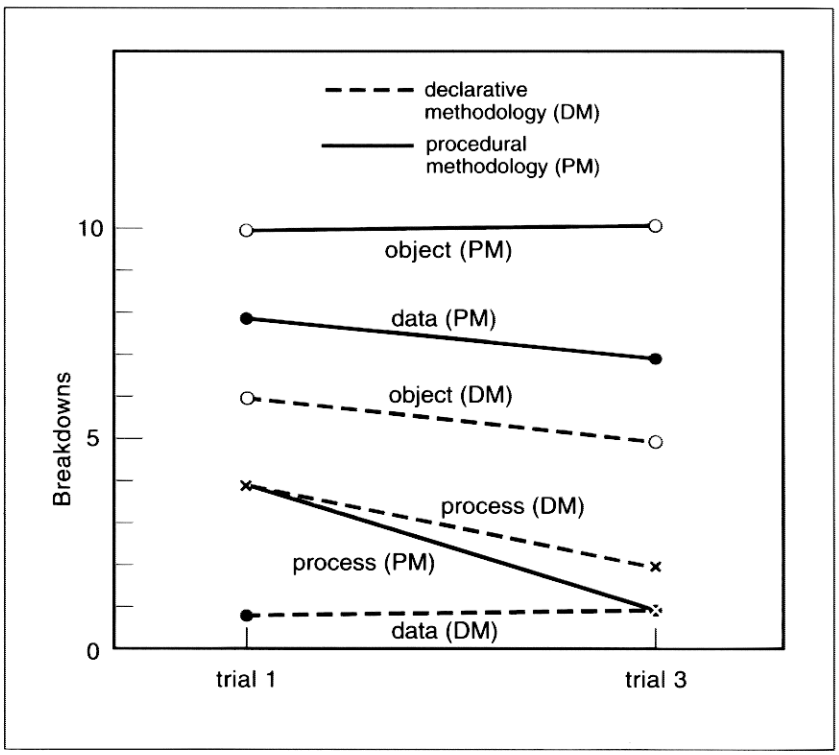

Figure 3. Unresolved declarative and procedural methodology breakdowns for each methodology from trial 1 to trial 3

On the other hand, those participants who made exclusive use of declarative methodology knowledge had a total of 46/1/45 methodology breakdowns. Table 6 summarizes the results.[7]

Furthermore, the number of methodology breakdowns committed by the declarative methodology group on the first and third trials was 24/0/24 and 22/1/21, respectively. The corresponding figures for the procedural methodology group were 17/8/9 and 7/2/5, respectively. Hence, the use of procedural knowledge is negatively associated with the number of methodology breakdowns committed initially as well as positively associated with the number of breakdowns resolved. Further, ability to specify information requirements over time improved only for those participants who acquired procedural methodology knowledge.

Examination of the different types of methodology breakdowns for the declarative methodology group shows that, on both trials, the majority of breakdowns were procedural in nature, 17/0/17 of 24/0/24 in trial 1 and 16/1/15 of 22/1/21 in trial 3. Corresponding figures for the procedural methodology group were 6/1/5 of 17/8/9 on trial 1 and 5/2/3 of 7/2/5 on trial 3. Interestingly, the procedural methodology group also demonstrated a greater ability to resolve declarative methodology breakdowns over time than the declarative methodology group. The declarative methodology group committed the same number of declarative methodology breakdowns (7/0/7 and 6/0/6 in trials 1 and 3, respectively) as the procedural methodology group (11/7/4 and 2/0/2, respectively), but was much less successful in resolving them. Further, note that the numbers of declarative methodology breakdowns committed by the procedural methodology group decreased over time (from 11 to 2), while those of the declarative methodology group were substantially similar (7 to 6). These findings suggest that those novice analysts who sought procedural knowledge did so when they already had a good grasp of the declarative aspects of the methodology.

In summary, the breakdown analyses for participants using a preponderance of either declarative or procedural methodology knowledge strongly suggest that novice analysts' problem solving was more effective when they sought procedural rather than declarative methodology knowledge. These findings support Anderson's learning theory [2].

Analysis of results. All the results presented here suggest that novice analysts found the process methodol-

[7]One participant, classified as a user of procedural methodology knowledge, used procedural knowledge predominantly in trial 1 and declarative knowledge predominantly in trial 3. In classifying this participant in the procedural methodology knowledge group, note that we are effectively biasing the results *against* finding the expected effects. An alternative view is that our results would have been even stronger had we excluded this participant from the procedural methodology group, since this participant had more unresolved breakdowns in trial 1 than in trial 3.

Table 5. Time spent in various activities during information requirements specification

	Means[1]					Performance over Time									
						Trial 1					Trial 3				
	Appln[2]	DecM	PrM	Soln	Totals	Appln	DecM	PrM	Soln	Totals	Appln	DecM	PrM	Soln	Totals
Process															
S4	21.4	.0	3.7	46.7	71.8	34.2	.0	7.4	76.0	117.6	8.6	.1	.0	17.3	26.0
S11	22.3	.5	4.0	47.8	74.1	24.2	.0	7.1	55.8	87.0	20.4	1.0	.0	39.7	61.1
Mean	21.8	.3	3.9	47.2	72.9	29.2	.0	7.3	65.9	102.3	14.5	.5	.0	28.5	43.5
%	29.9	.3	5.3	64.7	100.3	28.5	.0	7.1	64.4	100.0	33.3	1.1	.0	65.5	99.9
Data															
S6	29.4	1.8	1.6	51.1	83.5	38.1	1.1	3.0	65.8	108.0	20.6	2.5	.3	36.4	59.0
S10	28.6	3.8	.0	50.5	82.9	28.3	6.3	.0	56.8	91.4	28.9	1.3	.0	44.3	74.4
Mean	29.0	2.8	.8	50.8	83.2	33.2	3.7	1.5	61.3	99.7	24.7	1.9	.1	40.3	66.7
%	34.8	3.4	1.0	61.1	100.2	33.3	3.7	1.5	61.5	100.0	37.0	2.8	.1	60.3	100.2
Object															
S3	30.8	6.1	.0	92.4	129.5	43.1	9.1	.0	100.9	153.1	18.5	3.1	.0	83.8	105.8
S12	20.9	7.5	.0	35.0	63.4	18.2	11.1	.0	36.6	65.8	23.6	4.0	.0	33.3	60.9
Mean	25.8	6.9	.0	63.7	96.4	30.6	10.1	.0	68.8	109.5	21.1	3.7	.0	58.6	83.4
%	26.8	7.2	.0	66.0	100.0	28.0	9.2	.0	62.8	100.0	25.3	4.5	.0	70.3	100.1
Overall															
Mean	25.5	3.3	1.6	53.9	84.2	31.0	4.6	2.9	65.3	103.8	20.1	2.0	.0	42.5	64.5
%	30.3	3.9	1.9	64.0	100.1	29.9	4.4	2.8	62.9	100.0	31.2	3.1	.0	65.9	100.2

[1]*means except for "Totals" columns*
[2]*The documents referenced are:*
Appln=application
DecM=declarative methodology knowledge
PrM =example solution using given methodology (procedural methodology knowledge)
Soln =the problem solver's solution

Table 6. Effect of declarative and procedural methodology knowledge in information requirements specification

	Overall		Learning			
			Trial 1		Trial 3	
	DecMeth[1]	PrMeth	DecMeth	PrMeth	DecMeth	PrMeth
Declarative Methodology Group (S3, S10, S12)						
Total times (minutes)	34.9	.0	26.5	.0	8.4	.0
Breakdowns[2]	13/0/13 (46/1/45)	33/1/32	7/0/7 (24/0/24)	17/0/17	6/0/6 (22/1/21)	16/1/15
Procedural Methodology Group (S4, S6, S11)						
Total times (minutes)	4.7	17.8	1.1	17.5	3.6	.3
Breakdowns	13/7/6 (24/10/14)	11/3/8	11/7/4 (17/8/9)	6/1/5	2/0/2 (7/2/5)	5/2/3

[1] *DecMeth=declarative methodology*
PrMeth =example solution using given methodology (procedural methodology knowledge)
[2] *The breakdown entries are:*
1. the number of breakdowns
2. the number of breakdowns subjects resolved
3. the number of unresolved breakdowns

ogy easier to use than the data methodology, which, in turn, was easier to use than the object methodology. We present the thoughts of some of the participants, reported at the debriefing sessions. These thoughts illustrate very well the perceived differences in using the methodologies.

Subject S11, who used the process approach, states:

> I prefer the data flow process method. I think in processes. I can envision the data changing. There are times when I can see object-oriented would be really superior, but I think in processes.

Although S12 used the object methodology, he, also, indicates a clear preference for process methods:

> I haven't taken systems analysis yet, but . . . my preference is data flow diagrams; it is easier. I have a hard time with object-oriented; it's very difficult to me . . . what is calling and called object.

Similarly, S10, who used the data methodology, finds the process approach easier to apply. He states:

> Most people would have problems with JSD unless they had previous implementation experience . . . it's a really different approach than structured systems analysis.

Subject S4, who had no difficulty in applying the process methodology, illustrates the ease of learning to use that methodology. She states:

> The only way I know is what I learned in this class . . . I knew what was expected as far as output (documentation for the process methodology) and by then (trial 3), I was obviously more comfortable with the method because I didn't check back at the sample case as much.

Hence, the participants themselves are extremely articulate about the differences in their ability in learning to use the three methodologies. They would almost unanimously prefer to use the process method rather than JSD or object-oriented methods.

Examining the literature reveals substantial support for the notion that novice problem solvers prefer to use processing or procedural approaches to solve problems. For example, Papert [17] claims that programming procedurally fits novice's preexisting cognitive notions. Soloway et al. [22] showed that students write correct equations more often when solving word problems with procedural languages than with algebra, which is nonprocedural. Similarly, Welty and Stemple [23] found that student programmers using a procedural language outperformed programmers using a nonprocedural language. In a study on software design, Ratcliffe and Siddiqi [19] found that students preferred to use a process approach, although Jackson's data structure approach [11] would have led unequivocally to a correct solution. Their results persisted even with students who had been trained in Jackson's data structure approach. Hence, there is substantial evidence to suggest that novices find process approaches more natural than approaches based on data. Pennington [18], based on empirical evidence, also suggests that procedural approaches may be more natural than other approaches for experienced programmers.

Discussion

Our research investigated the effect of three methodologies on novice analysts' performance in learning to specify information requirements. The findings and the limitations of the study are discussed along with implications of the results for researchers, instructors in IS, and practitioners.

The results demonstrate that, of the three methodologies investigated, novice analysts were better able to apply the process methodology and least able to apply the object methodology. Further, significant learning over three trials occurred only for the process methodology. These results are substantiated by the verbal state-

ments of the participants following the experimental session.

There are two possible reasons for the significantly better performance of process over data and object methods. The first reason is that process methods may be better defined. They were developed earlier than both data and object methods (see, for example, [3], [6], and [10]). That process methods may be better defined is supported by the fact that the structured techniques are those most frequently embedded in CASE tools. Further evidence that the more recent methodologies may be less well defined can be seen in the need to modify the object approach to include an explicit step to guide participants where none was provided by the methodology (see "Methodologies" section).

The second reason for the superior performance of participants using the process approach compared with those using the data and the object approaches may be that (novice) problem solvers perform better with process or procedural methods. Substantial evidence can be found to support this notion in both the cognitive psychology literature per se and in that on computer programming and design.

The potential limitations of this study center on the small number of participants, the quality of training in the methodologies, and on the use of novice systems analysts. First, a small number of participants is the norm for exploratory studies such as this. For example, protocol studies by Jeffries et al. [12], Guindon et al. [9], Kant and Newell [13], and Adelson and Soloway [1] in the related area of systems design collected data from nine, eight, two, and five subjects, respectively. Most of the studies examined only a fraction of those protocols in detail, however, and none of them formally coded their data. We attempted to overcome the problems associated with small samples in two ways: 1) we used a repeated-measures design and 2) we used diverse types of analyses of the verbal protocol data. In particular, the data were coded to reflect the time spent referencing experimental materials and the breakdowns that occurred in problem solving. Further, intuitive support for the findings also comes from novice analysts' comments on their own learning processes.

Second, there were significant differences in the subjects' ability to use the three methodologies. As evidence of our attempts to control participants' initial methodology knowledge, we cite the following measures taken: 1) the data and object methodologies were modified to bridge certain areas in which students experienced difficulties in pilot tests, 2) training was achieved using standard instructional materials for both the initial class session and for the review the participants received at the beginning of the study session, and 3) the students were given equivalent class and homework assignments for each methodology. Recall, also, that students' perceptions of their ability to apply the given methodology was captured in a background questionnaire. Subjects' perceived abilities to apply the object methodology were only slightly lower than their perceived abilities to apply the process and data methodologies (see "Method" section). We believe, therefore, that the observed differences in performance using the three methodologies reflect true differences in applicability of the methodologies themselves.

Third, since our research investigated learning, we used novice analysts as subjects (see "The Study" section). Our results do not, therefore, generalize to the body of practicing systems analysts.

Implications of the Results

What, then, are the implications of the results for researchers, instructors in information systems, and practitioners?

From the viewpoint of *researchers*, the findings of this study should first be validated using other research methodologies, due to the use of the small sample necessitated by process-tracing studies. Second, similar studies should be undertaken with experienced systems analysts to assess their performance in learning to use each of the methodologies. Third, this study addressed only the ability to learn the three methodologies. Clearly, research is needed that addresses the performance of mature users of each of the methodologies. Fourth, and perhaps most important, based on the specific objectives of this study, our results represent a challenge to methodologists, particularly to supporters of the data and the object approaches. Perhaps those methodologies need to be better defined to facilitate learning to use them in practice. In particular, our results suggest that methodologies must provide clear, unambiguous procedural details on how to apply them. Object-oriented requirements methods, in particular, have proliferated since this research was performed. Some of them, such as Coad and Yourdon [4], contain much more detailed techniques for specifying IS requirements than the Booch approach used in this study. There has not been more recent material addressing the application of JSD, however.

From the viewpoint of *instructors* of systems analysis and design who may be interested in introducing object-oriented techniques into their curricula, it appears it may be considerably more difficult to teach students to use the object-oriented approach to systems development than the more traditional structured techniques. More specifically, the difficulties encountered by students in applying the object methodology in this research suggest that more procedural training in the form of example solutions and practice exercises will be required for this methodology.

From the viewpoint of *practitioners* choosing methodologies, the results of this study suggest that object-oriented approaches may not be best for all problem solvers in all situations. IS managers need to evaluate the claims made by devotees of object-oriented approaches very carefully. Further, given that novices who have no preconceived notions of structuring have problems learning object-oriented approaches, it is likely that experienced practitioners already comfortable with an alternative approach may fare even more poorly. Caution should be exercised in mandating the use of a new methodology or technique unless its performance has been demonstrated unequivocally.

A further issue for both researchers and practitioners that should be

addressed in the future is whether the applicability of a particular methodology depends on the nature of the problem to be solved. There is very little explicit consideration in the literature of the relationship between methodologies and application domains. Proponents of the object-oriented approach, for example, frequently claim it is a universally superior paradigm. However, its performance in different circumstances has not been evaluated. Further research and feedback from practice are needed in this area. ◻

Acknowledgments

The authors are indebted to Robert L. Glass for comments on an earlier version of this article. We also wish to acknowledge the assistance of Umesh Bhatia at Pennsylvania State University and Alex Heslin at Georgia State University. The Center for Interdisciplinary Research in Information Systems in the Smeal College of Business Administration at Pennsylvania State University and the College of Business at Georgia State University provided financial assistance.

References

1. Adelson, B. and Soloway, E. The role of domain experience in software design. *IEEE Trans. Softw. Eng. SE-11*, 11 (1985), 1351–1360.
2. Anderson, J.R. Acquisition of cognitive skill. *Psychol. Rev. 89*, 4 (1982), 369–406.
3. Booch, G. *Software Engineering with Ada*. Second ed. Benjamin/Cummings, Menlo Park, Calif., 1987.
4. Coad, P. and Yourdon, E. *Object-Oriented Analysis*. Second ed. Yourdon Press/Prentice-Hall, Englewood Cliffs, N.J., 1991.
5. Davis, G.B. Strategies for information requirements determination. *IBM Syst. J. 21*, 1 (1982), 4–30.
6. De Marco, T. *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
7. Ericsson, K.A. and Simon, H.A. *Protocol Analysis: Verbal Reports as Data*. The MIT Press, Cambridge, Mass., 1984.
8. Gane, C. and Sarson, T. *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall, Englewood Cliffs, N.J., 1979.
9. Guindon, R., Krasner, H., and Curtis, B. Breakdowns and processes during the early activities of software design by professionals. In *Empirical Studies of Programmers: Second Workshop*, 1987.
10. Jackson, M. *System Development*. Prentice-Hall, Englewood Cliffs, N.J., 1983.
11. Jackson, M.A. *Principles of Program Design*. Academic Press, London, 1975.
12. Jeffries, R., Turner, A., Polson, P., and Atwood, M. The process involved in designing software. In *Cognitive Skills and Their Acquisition*, J.R. Anderson, Ed.
13. Kant, E. and Newell, A. Problem solving techniques for the design of algorithms. *Inf. Process. Manag. 28*, 1 (1984), 97–118.
14. Miller, G.A. The magical number seven, plus or minus two. *Psychol. Rev. 63*, 1 (1956), 81–97.
15. Necco, C.R., Gordon, C.L., and Tsai, N.W. Systems analysis and design: Current practices. *MIS Q. 11* (1987), 461–475.
16. Olle, T.W., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Van Assche, F., and Verrijn-Stuart, A. *Information Systems Development Methodologies*. Second ed. Addison-Wesley, Reading, Mass., 1991.
17. Papert, S. *Mindstorms, Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.
18. Pennington, N. Stimulus structures and mental representations in expert comprehension of computer programs. *Cog. Psychol. 19* (1987), 295–341.
19. Ratcliffe, B. and Siddiqi, J. An empirical investigation into problem decomposition strategies used in program design. *Int. J. Man-Machine Stud. 22*, 1 (1985), 77–90.
20. Ross, D.T. and Schoman, K.E. Structured analysis for requirements definition. *IEEE Trans Softw. Eng. SE-3*, 1 (1977), 6–15.
21. Simon, H.A. The architecture of complexity. *In Proceedings of the American Philosophical Society 106* (1962), 467–482.
22. Soloway, E., Lochhead, J., and Clement, J. Does computer programming enhance problem-solving ability? Some positive evidence on algebra word problems. In *Computer Literacy*, R. Seidel, Ed. Academic Press, New York, 1983.
23. Welty, C. and Stemple, D.W. Human factors comparison of a procedural and a nonprocedural query language. *ACM Trans. Database Syst. 6*, 4 (1981), 626–649.
24. Yadav, S.B., Bravocco, R.R., Chatfield, A.T., and Rajkumar, T.M. Comparison of analysis techniques for information requirement determination. *Commun. ACM 31*, 9 (1988), 1090–1097.
25. Yourdon, E. *Modern Structured Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1989.

About the Authors:

IRIS VESSEY is an associate professor of management information systems at The Pennsylvania State University. Research interests focus on the cognitive processes underlying the analysis and design of software—the cognitive fit of representations, methodologies, and techniques to software tasks, the role of the application in the selection of approaches to systems development, and the evaluation of emerging information systems technologies. **Author's present address:** Department of Management Science and Information Systems, The Pennsylvania State University, University Park, PA 16802; email: ixvl@psuvm.psuvm.edu

SUE CONGER has been on the faculties of Baruch College, CUNY, Georgia State University, and New York University. Current research interests are ethics and IT, software engineering, and innovative use IT. She has a consulting practice, which draws on her work experiences in the information systems field. **Author's Present Address:** email: sconger@aol.com; fax: (214) 931-9125