

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313: 761-4700 800: 521-0600



Order Number 9123489

**Information system development and the use of electronic
meeting systems**

Liu, Kung-Chao, Ph.D.

The University of Arizona, 1991

Copyright ©1991 by Liu, Kung-Chao. All rights reserved.

U·M·I

**300 N. Zeeb Rd.
Ann Arbor, MI 48106**

NOTE TO USERS

**THE ORIGINAL DOCUMENT RECEIVED BY U.M.I. CONTAINED PAGES
WITH SLANTED PRINT. PAGES WERE FILMED AS RECEIVED.**

THIS REPRODUCTION IS THE BEST AVAILABLE COPY.



INFORMATION SYSTEM DEVELOPMENT AND THE USE OF
ELECTRONIC MEETING SYSTEMS

by

Kung-Chao Liu

Copyright © Kung-Chao Liu 1991

A Dissertation Submitted to the Faculty of the
COMMITTEE ON BUSINESS ADMINISTRATION
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

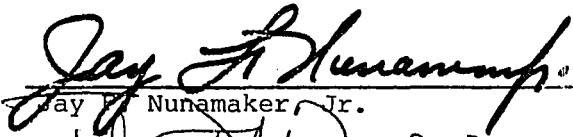
1 9 9 1

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

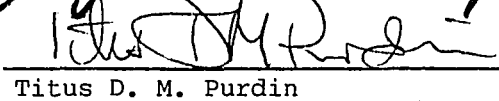
As members of the Final Examination Committee, we certify that we have read
the dissertation prepared by Kung-Chao Liu

entitled Information System Development and the Use of Electronic
Meeting Systems

and recommend that it be accepted as fulfilling the dissertation requirement
for the Degree of Doctor of Philosophy.


Jay F. Nunamaker, Jr.

2/22/91
Date


Titus D. M. Purdin

2/22/91
Date


Olivia R. Liu Sheng

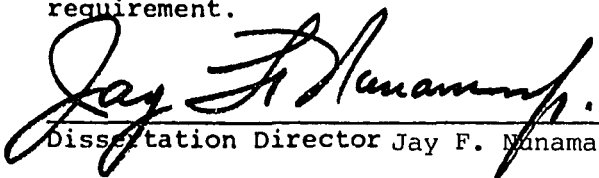
2/22/91
Date

Date

Date

Final approval and acceptance of this dissertation is contingent upon the
candidate's submission of the final copy of the dissertation to the Graduate
College.

I hereby certify that I have read this dissertation prepared under my
direction and recommend that it be accepted as fulfilling the dissertation
requirement.


Dissertation Director Jay F. Nunamaker, Jr.

2/22/91
Date

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: Wang Chao Liu

ACKNOWLEDGMENTS

I am grateful to the members of my dissertation committee. Dr. Jay Nunamaker provided the positive research environment that made the work on this dissertation possible. Dr. Titus Purdin devoted considerable time to both the content and form of this dissertation. Dr. Olivia Sheng gave me advice that helped me persevere in my doctoral program. Dr. Bernard Zeigler and Dr. Jerzy Rozenblit taught me ways of solving problems that were helpful in conducting this research.

My colleagues and friends—especially Josh Chao, Alan Dennis, Jimmy Ferng, Glenda Hayes, Bruce Herniter, Simon Leung, Bill Saints, Sun Wu—have always been available to give me a hand whenever I needed it.

My parents, Dr. and Mrs. Chih-Wan Liu, strongly believe in the value of education. Throughout my life, they have always done their best to provide me the best education. In the past three years, my wife Hueili has accompanied me across all the hurdles that appeared one after another. Their love and support have enabled me to arrive this far.

TABLE OF CONTENTS

LIST OF FIGURES	9
ABSTRACT	10
1 INTRODUCTION	12
1.1 Issues of Information System Development Research	12
1.2 Framework of the Research	15
1.3 Organization of This Dissertation	17
2 INFORMATION SYSTEM DEVELOPMENT	20
2.1 Problems of Previous Models	20
2.2 The Generational Framework	22
2.2.1 The Process	22
2.2.2 The Object	25
2.2.3 The Framework Summarized	28
2.3 Discussions of the Generational Framework	28
2.3.1 An Interpretation of the Generational Framework	28
2.3.2 Comparisons to Existing Models	30
2.3.3 An Application of the Generational Framework	32
2.4 Concluding Remarks	34
3 SYSTEM REQUIREMENTS DETERMINATION	35
3.1 Requirements Determination Defined	35

3.2	Requirements Elicitation	38
3.2.1	Problem-solving Process	38
3.2.2	Engineering Design Process	42
3.2.3	Requirements Elicitation Process	45
3.3	Requirements Specification	49
3.3.1	Requirements Specification Problem	49
3.3.2	Problem Structure	50
3.3.3	Solution Structure	53
3.4	Concluding Remarks	54
4	REQUIREMENTS DETERMINATION BY GROUP	56
4.1	The Problem of Participation	56
4.2	The Group Approach to Requirements Determination	60
4.3	Composition of Group	62
4.4	Concluding Remarks	63
5	FITTING COMPUTER AIDS	65
5.1	The Approach to Fitting Computer Aids	65
5.2	The Usability Problem	68
5.3	The Configuration Problem	73
5.3.1	The Problem of Deriving Subconfigurations	74
5.3.2	The Problem of Deriving Total Configurations	75
5.4	Approximation Methods for Deriving Configurations	76
5.4.1	Approximation Method for Deriving Subconfigurations	76
5.4.2	Approximation Method for Deriving Total Configurations	79
5.4.3	Remarks on the Approximation Methods	81
5.5	Concluding Remarks	82

6	FITTING ELECTRONIC MEETING SYSTEMS	84
6.1	Classification of Information Types	84
6.2	Information Support in Requirements Determination	88
6.3	GroupSystems as Computer Aids	92
6.4	The Use of GroupSystems in Requirements Determination	98
6.4.1	The Usability Problem	98
6.4.2	The Configuration Problem	101
6.5	Concluding Remarks	105
7	USING GROUPSYSTEMS IN REQUIREMENTS DETERMI-	
	NATION	107
7.1	Background of the Case	107
7.2	Analysis of the Meeting Sessions	109
7.3	Concluding Remarks	113
8	CONCLUSIONS	114
8.1	Summary	114
8.2	Future Research	117
	REFERENCES	118

LIST OF FIGURES

1.1 Divisions of this dissertation and their relations	18
2.1 Process part of the generational framework	23
2.2 The process and the object of the generational framework	26
2.3 Generations of the generational framework	27
2.4 Discontinuous progress of system development	29
2.5 Evolution of systems	33
3.1 The problem and solution structures for requirements determination	37
3.2 The elements of requirements determination	47
3.3 Example intent structures	52
5.1 Example mappings between task aspects, information types, and computer aids functionalities	67
5.2 Example mappings relevant to one task aspect	70
5.3 Example mappings relevant to one computer aids functionality . . .	72
5.4 Demonstration of the subconfiguration method	78
5.5 Demonstration of the total configuration method	80
6.1 Granularity and resolution of information types classification	86
6.2 The mapping between task aspects and information types	90
6.3 The mapping between information types and GroupSystems tools .	94
6.4 The extended mapping between information types and GroupSys- tems tools	102

6.5	Configurations of GroupSystems tools	104
7.1	Tools used in the meeting sessions of the case	110
7.2	Information types involved in the meeting sessions of the case . . .	112

ABSTRACT

Information system requirements determination is a key area in management information systems research that includes the problems of requirements specification, requirements elicitation, and user involvement. The combination of these three problems is a research area which we call the group approach to information system requirements determination. The main contribution of this research is a model to be used for the problem of fitting existing computer aids to this research area and a set of methods for solving the usability and configuration problems when using such computer aids. The usability problem is that of determining whether a set of computer aids can be used effectively in accomplishing the task of requirements determination. The configuration problem is that of selecting a minimum collection of functionalities necessary for economically supporting all aspects of requirements determination.

Electronic meeting systems are the general category of computer aids that we are interested in applying to the task. In particular, the GroupSystems electronic meeting system developed at The University of Arizona is used as a case in this research. Characteristics of the requirements determination task and profiles of GroupSystems tools are combined into our model for fitting computer aids to a given task. We then derive the answers regarding the usability and configuration of GroupSystems in the group approach to information system requirements determination. We also compare the derived configurations to the GroupSystems tools used in an authentic case.

The main points of this research include: (1) proposal of the concept of

fitting computer aids to, instead of developing new computer aids for, a task area; (2) proposal of a model for fitting computer aids via a classification of information types; (3) analysis of the natures of the fitting model and the usability and configuration problems; (4) proposal of approximation methods for solving the configuration problem; (5) analysis of the task area — the group approach to information system requirements determination; (6) analysis of the use of GroupSystems in the task area by applying the proposed fitting model and approximation methods; and (7) demonstration of the usefulness of the fitting model and approximation methods by analyzing an authentic case of using GroupSystems tools.

CHAPTER 1

INTRODUCTION

1.1 Issues of Information System Development Research

One major category of research in the management information systems (MIS) field has been the development of information systems. Information system development is concerned with the production of information systems within the constraints imposed by the environments where the information systems will operate. During the course of developing information systems, the determination of information system requirements has been considered the most difficult and important part of the development process. It is difficult because the parameters associated with a particular environment are difficult to acquire. It is important because the results accumulated during requirements determination will have fundamental effects on the remaining phases of the development process.

The problem of determining requirements has two parts: the elicitation problem and the specification problem (Friedman & Cornford, 1989). The former concerns the identification of user requirements and the latter is about the assurance of the accuracy of the requirements specification. The problem of assurance is to find a systematic representation of requirements that will allow information system developers and users to judge whether a set of requirements is consistent, accurate, and complete. During the 1960s and 1970s, the search for an ultimate representation method dominated the research of requirements determination methods. The problem of elicitation was assumed to be solvable by accurate

or efficient methods of representation in the sense that, with an unambiguous representation method, developers could easily define a set of correct requirements for users (Friedman & Cornford, 1989; Dennis, George, Jessup, Nunamaker, & Vogel, 1988).

Since the 1980s, researchers have recognized the imbalance of the elicitation and the specification sides of the requirements determination problem and have begun to explore the gap between them. This gap is usually signified by the fact that users cannot express their requirements explicitly and developers have no good way to accurately obtain users' requirements. The solution to this difficulty lies in better methods of elicitation that incorporate higher degrees of user involvement. It has been suggested that users should be involved throughout a development process in which decisions are made in consensus manner by developers as well as users (Land, 1982). Trade journals began to publicize this consensus approach to system development (see, for example, Kull, 1987) in the mid-1980s. The results of academic studies about user involvement and the success of the consensus approach were varied (Ives & Olson, 1984) and evidence presented by industry regarding the effectiveness of the consensus approach to system development are still inconclusive.

The use of computer aids to assist in the development of information systems has been called computer-aided software engineering (CASE) since the mid-1980s. Virtually all current CASE products for use in information system requirements determination emphasize representation methods (see, for example, those reviewed by *Byte* staff, 1989, and *Spectrum* staff, 1990). The essential difference between these products and their precursors like PSL/PSA (Teichroew & Hershey, 1977) is that they use graphical representations instead of textual representations. Since computer aids have done well in the specification side of requirements de-

termination, the conjecture is that computer aids can also assist in requirements elicitation, particularly when coupled with a high degree of user involvement—namely, the group approach to information system requirements determination.

The research in information system development at The University of Arizona assumed the representation-based PSL/PSA approach in the 1970s and gravitated to the PLEXSYS approach (Konsynski, Kottemann, Nunamaker, & Stott, 1984–1985) in the early 1980s. The PLEXSYS approach provided system developers an integrated workbench equipped with various tools. This approach, however, was still biased toward developers. The workbench approach evolved into the electronic meeting system approach (Dennis et al., 1988; Vogel, Nunamaker, George, & Dennis, 1988) during the late 1980s. The electronic meeting system approach points to a way that could lead to computer aids for use in a group approach to information system requirements determination because electronic meeting systems provide facilities that support the interaction process of group meetings. This computer-aided process of group interaction shows promise for assisting in conducting requirements elicitation sessions in which users are involved.

In summary, information system requirements determination is a key area in MIS research that includes not only the problem of requirements specification but also the problems of requirements elicitation and user involvement. The combination of these three problems is a research area which we call the *group approach to information system requirements determination*. We want to use computer aids to assist in this requirements determination task performed by a group. In order to solve the requirements determination problem as a whole, a computer aid should take into consideration all the representation, elicitation, and user involvement aspects.

1.2 Framework of the Research

This research is about the use of computer aids to facilitate a group approach to information system requirements determination. Instead of designing an entirely new set of computer aids specifically tailored to requirements determination, our approach is to investigate the feasibility of applying existing computer aids to the problem. Consequently, the main concern of this research is to propose a model for fitting existing computer aids to this task and a set of methods for solving the usability and configuration problems of the computer aids with respect to the task. The usability problem is that of determining whether the computer aids under study can effectively assist in supporting group oriented requirements determination. In other words, we want to determine whether the functionalities of a set of existing computer aids are sufficient to support all aspects of the requirements determination task and whether each functionality of these computer aids can be used to support particular aspects of the task. The configuration problem is that of selecting a minimum collection of the functionalities among the computer aids necessary for economically supporting all aspects of the task.

We proceed by first analyzing our research area—the group approach to information system requirements determination. We analyze the area in layers, from general to specific. We first investigate the characteristics of the task of information system development in general, then narrow the study to the task of information system requirements determination, and finally examine the requirements determination task performed by a group of people in a consensus manner. We propose in each layer a framework for describing the process of elicitation and the representation of specifications. All the frameworks are related to each other and form a complete analysis of the research area.

Based on the concept that computer aids have to fit the task, we propose a model for matching functionalities of computer aids to the aspects of a task area. This model uses a classification of information types to mediate the task aspects and the functionalities of computer aids in the sense that the functionalities of computer aids can provide some types of information and that the task aspects should be supported by certain types of information. The model can then be used to answer the usability and configuration problems of computer aids with respect to the task. The usability problem is divided into four subproblems. We devise methods for solving each of them. The configuration problem is decomposed into two subproblems. We analyze the intractable natures of these subproblems in detail and design approximation methods for coping with them.

Electronic meeting systems are the general category of computer aids that we are interested in applying to the requirements determination task. In particular, the GroupSystems electronic meeting system developed at The University of Arizona is used as a case in this research. Profiles of GroupSystems tools, a list of information types, and the characteristics of the task, which result from our analysis of the research area, are arranged into our model for fitting computer aids to a given task. We then derive the answers regarding the usability and configuration of GroupSystems electronic meeting system in the group approach to information system requirements determination. We also compare the derived configurations to the GroupSystems tools used in an authentic case of using GroupSystems tools in requirements determination.

To summarize, the main points of this research include:

1. proposal of the concept of fitting computer aids to, instead of developing new computer aids for, a task area;

2. proposal of a model for fitting computer aids via a classification of information types;
3. analysis of the natures of the fitting model and the related usability and configuration problems;
4. proposal of approximation methods for solving the configuration problem;
5. analysis of the task area—the group approach to information system requirements determination;
6. analysis of the use of GroupSystems electronic meeting system in the task area by applying the proposed fitting model and approximation methods; and
7. demonstration of the usefulness of the fitting model and approximation methods by analyzing an authentic case of using GroupSystems tools.

1.3 Organization of This Dissertation

This dissertation contains eight chapters. Contents of the chapters and the lines of reasoning between the chapters are shown in Figure 1.1. Each chapter is represented as a box with the chapter number labeled on the upper-left corner. The box for Chapter 5 is divided into three parts to show further details.

Chapters 2, 3, and 4 are concerned with the analysis of the research area, namely, the group approach to requirements determination. Chapter 2 is about information system development in general; Chapter 3 is about information system requirements determination; and Chapter 4 is about the group approach to information system requirements determination. Chapter 5 contains three topics: the model for fitting computer aids, the analysis of the usability problem and the methods for solving the usability problem, and the analysis of the configuration problem

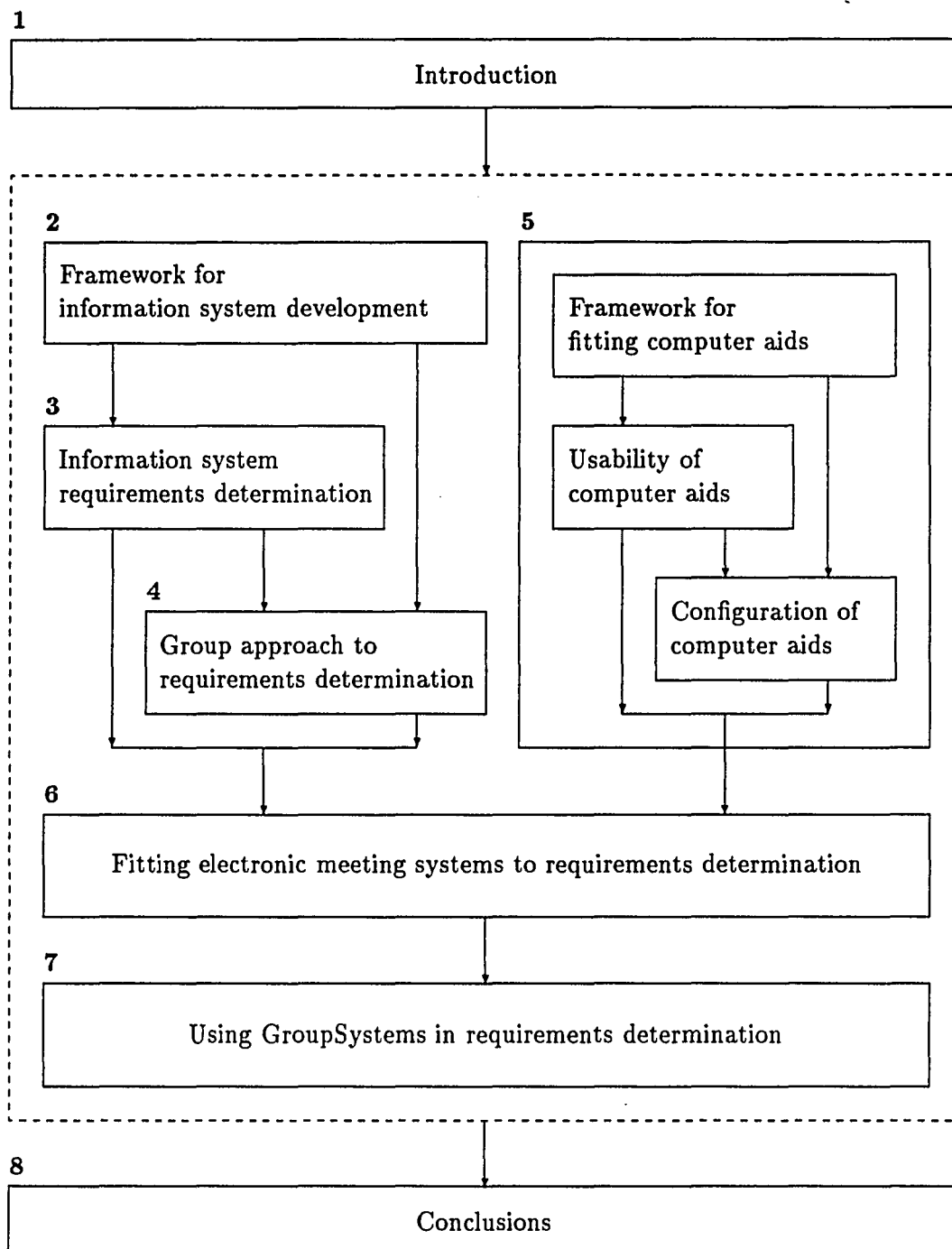


Figure 1.1: Divisions of this dissertation and their relations.

and the approximation methods for solving the configuration problem. In Chapter 6, the fitting model and the methods for solving the usability and configuration problems are applied to GroupSystems tools and the area of the group approach to requirements determination. An authentic case of using GroupSystems tools in requirements determination is analyzed in Chapter 7. Finally, Chapter 8 presents the conclusions.

CHAPTER 2

INFORMATION SYSTEM DEVELOPMENT

The main focus of this dissertation is the requirements determination problem, which includes the requirements elicitation and specification tasks. Since these two tasks are part of the larger information system development process, we shall not discuss them without first investigating the overall development process. That is, to adequately study the requirements elicitation and specification tasks, we shall first study their context—the information system development process.

In this chapter, we examine the problems of previously proposed development process models and, based on our observations, propose a generational framework for information system development. We choose to use the term *generational* because we view both the development process and the object to be developed as being spawned in generations. We also offer an interpretation of the generational framework, compare the generational framework with previous models, and present an application of the generational framework.

2.1 Problems of Previous Models

Most of the information system development process models proposed to date assume that the development process consists of a sequence of somewhat independent activities each having specific products and goals associated with them. Further, they assume that those products serve as the interface between the activities and as the gauge of the progress of development (Harandi, 1988).

This linear progression of activities presents a view of managerial control (Balzer, 1985) that does not necessarily reflect what is really happening when developers are doing their jobs.

In the interest of better understanding of the development work, the following facts should be taken into consideration:

- A process model should consist of development activities, instead of activities for managerial control.
- Customers should interact with developers (in, for example, the manner described in Carroll, Thomas, & Malhotra, 1979). A balanced consumer/supplier relation between customers and developers fosters productive interactions between them. Among other things, the interactions help developers and customers keep a product dynamic and determine when it is obsolete.
- The development process should not be a linear progression because in most cases full system requirements can rarely be adequately stated in advance.
- The development process is an intertwining process in that the specified requirements may have to be changed as a result of development efforts. For example, limitations of available implementation technology may force a specification change or implementation choices may suggest augmentations to a specification (Swartout & Balzer, 1982).
- It is unrealistic to treat maintenance separately from development. It is also implausible to perform maintenance on source code without paying attention to the intellectual activities leading to such work.
- Previously developed versions of specifications and product should be available for reference during the development process.

These observations have motivated the proposal of a generational framework that can describe information system development activities more accurately. The generational framework helps set up the context in which the requirements elicitation and specification problems are studied in the following chapters.

2.2 The Generational Framework

2.2.1 The Process

The generational framework for information system development consists of two closely related parts: the development process and the object developed during the process. A generation of the development process consists of four stages: elicitation, specification, elaboration, and animation (Figure 2.1).

- *Elicitation* is concerned with acquiring from the customer all the facts related to the required system.
- *Specification* is concerned with formalizing and specifying the system requirements in a clear and consistent manner.
- *Elaboration* is concerned with realizing the requirements.
- *Animation* is concerned with demonstrating the realization. Here we regard animation as any means that can be used to interpret, validate, or illustrate the realization of the system for the customer, in a sense much broader than that in Cohen (1982) or Kramer and Ng (1988).

The active nature of the process is captured in the transitions from stage to stage: elicitation \longrightarrow specification \longrightarrow elaboration \longrightarrow animation. The two left-hand-side stages (elicitation and animation) of a generation in Figure 2.1 are customer-centered because customer's inputs are critical in these two stages. The

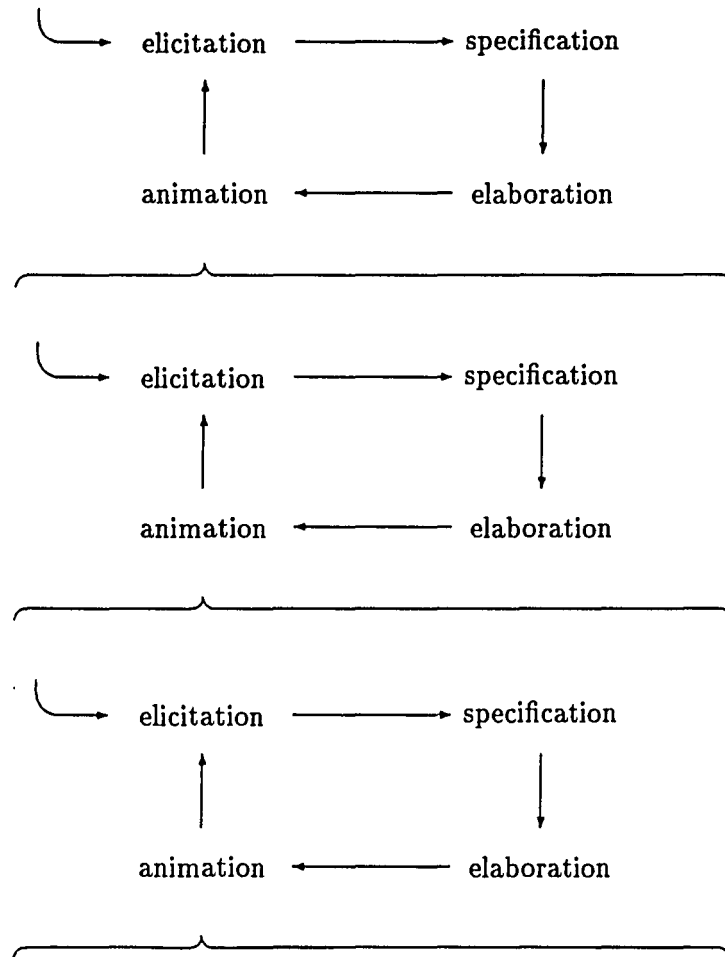


Figure 2.1: Process part of the generational framework.

two right-hand-side stages (specification and elaboration) are developer-centered because developer's effort is the locomotion in these two stages. This perspective emphasizes customer's involvement in the development process, which is the result of the consumer/supplier interaction between customer and developer.

From another perspective, the upper half of a generation in Figure 2.1 is the problem *representation process* that is concerned with constructing the problem part of the object. The lower half is the problem *solution process* that is concerned with constructing the solution part of the object.¹ The relationship between the problem- and the solution-orientations of the development process will become clearer when we discuss the object in the next subsection.

Both customer and developer are involved in all four stages of the process. Some stages will, however, be of more interest to one group than the other. The duration of a particular stage and the depth of customer/developer overlaps may vary considerably. After the animation stage, the customer and developer may decide to go back to the elicitation stage or to go on to a new generation of the four-stage process. The former case has been called *corrective* maintenance (Swanson, 1976), as its purpose is to remedy any discrepancies between the solution and the problem parts of the object. In the presence of *adaptive* or *perfective* maintenance (Swanson, 1976), the customer and developer should spawn a new generation of the development process.

In either case the concept of treating maintenance as *re-development* is observed: the maintenance begins with eliciting new requirements and the specifications and product (that is, the problem and solution parts of the object) together

¹The terms representation process and solution process are derived from the terms *problem representation* and *problem solution* (Voss & Post, 1988), which refer to the two parts of the course of problem-solving.

evolve into newer versions.

2.2.2 The Object

Parallel to the four-stage process is the object being worked on by the customer and developer. The object consists of a *problem structure* and a *solution structure* that are gradually built during the four stages. During the problem representation process of the first two stages, the problem structure is created. During the problem solution process of the later two stages, the solution structure is added and the relationships between components of the two structures are established. Figure 2.2 illustrates how the object is related to the two perspectives of the process mentioned in the previous subsection.

Davis (1988b) has identified that system requirements include descriptions of user needs, solution space (which is usually expressed in terms of the constraints present), and products behavior. These items make up the problem structure of the object. Other items also mentioned by Davis, such as architecture, module specifications, algorithms, and code, should go into the solution structure. Since we do not preclude any means of realization, the constitution of a solution structure might be different from the one portrayed here.

When a new generation of the process is spawned, a new version of the object is constructed and it becomes the focus of the new generation. Old versions of the object can subsequently be referenced but not modified. In this way, generations of the object parallel to generations of the process are formed. These generations constitute the history of development, as shown in Figure 2.3.

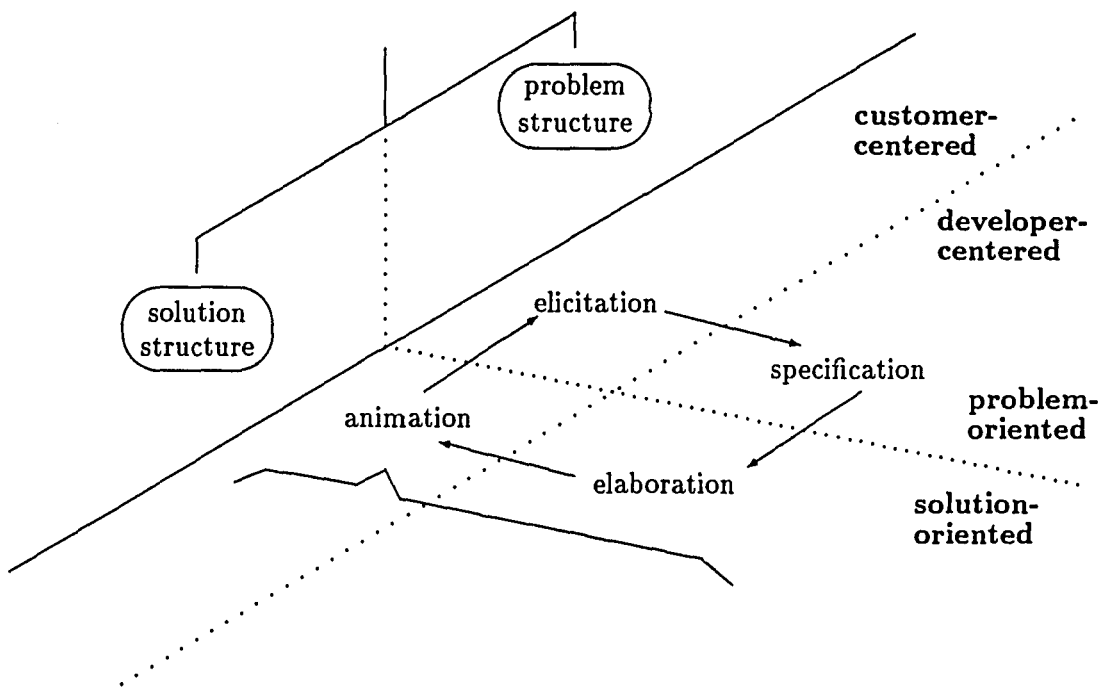


Figure 2.2: The process and the object of the generational framework.

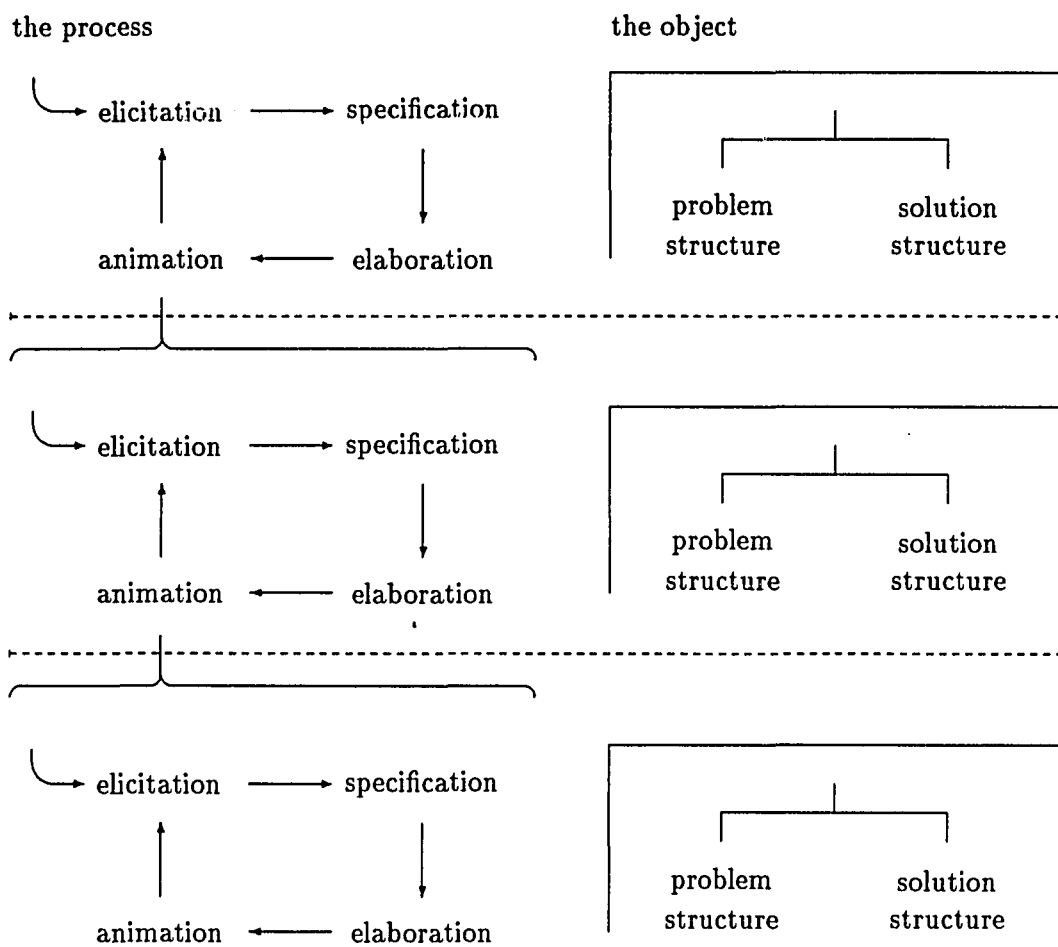


Figure 2.3: Generations of the generational framework.

2.2.3 The Framework Summarized

As illustrated in Figure 2.2, a generation of the framework consists of an instance of the object and an instance of the process that interact. Although involvement of the customer and the developer may vary from stage to stage, the process proceeds as a result of the cooperation between the two parties. They build up the problem structure of the object during the earlier two stages and the solution structure during the later two stages. As generations of the framework accumulate (Figure 2.3), a traceable history of the development efforts is chronicled. A version of the product is always available.

2.3 Discussions of the Generational Framework

2.3.1 An Interpretation of the Generational Framework

The progress of developing any system, as perceived by Lansdown (1987), is a discontinuous process illustrated in Figure 2.4. The curves indicate the process of development, which is the degree of completeness of the system at any particular time. Discontinuities occur when an existing set of ideas cease to have sufficient promise and new ideas are applied. We argue that this viewpoint of system development justifies the generational nature of the generational framework.

Each of the continuous curves between discontinuous points indicates a generation of the development process, while each discontinuous point indicates the time when a new generation of the system is spawned. This point of fracture is also a time when inventiveness happens. Referring to Figures 2.3 and 2.4, after cooperatively going through a generation of the development process, the customer and developer may initiate a new generation if they have new ideas. In this case, the degree of completeness of the system falls. After this point of fracture, they

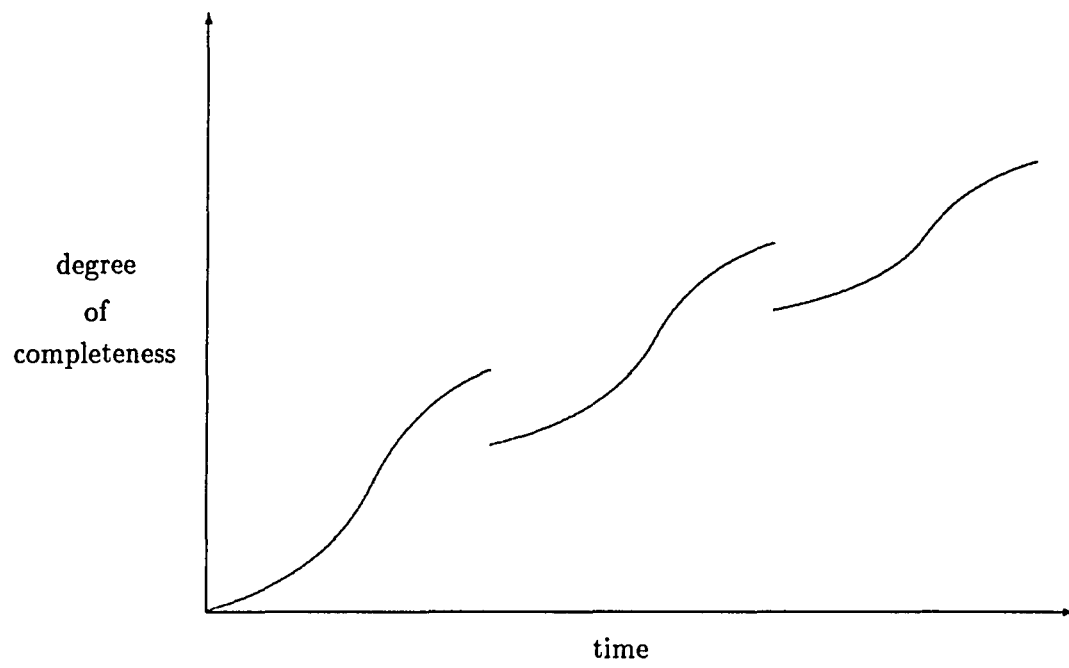


Figure 2.4: Discontinuous progress of system development.

work on the new ideas as a new generation of the development process. Each generation, it is expected, will contribute to the growing completeness of the target information system. It is the advent of the new ideas that boosts the degree of completeness of the system being developed.

2.3.2 Comparisons to Existing Models

The generational framework is generic and can be tailored to meet specific needs. The duration of each stage, the techniques used in each stage, and the representation structures of the object will vary. The result will be a number of specialized frameworks. An advantage of the generational framework, therefore, is that each generation may vary considerably from a previous one. Customer and developer can attempt their joint endeavor more flexibly.

Some previously proposed models, such as prototyping, may be regarded as instances of specialized frameworks. Here are how different categories of prototyping (Hekmatpour & Ince, 1988) are compared to the generational framework.

- *Incremental* prototyping is the identification of all requirements at once but implementing them gradually. This is a degenerate case of the framework that lasts only one generation.²
- *Throw-it-away* prototyping is a limited generational framework that concen-

²A similar degenerate case is Zmud's (1980) *evolutionary life cycle*, which consists of a requirements analysis phase and a series of development phases for versions of the product. Although this approach recognizes the evolutionary nature of the information system development work, it has two shortcomings. First, the process for developing each version (system design, program design, coding, testing, acceptance testing, and quality control testing) is not different from traditional development process. Second, the requirements analysis phase is a one-time work, which is not feasible.

trates on elicitation and animation stages of the process but does not handle the object explicitly.

- *Evolutionary* prototyping looks most similar to the generational framework. The process, however, stops and maintenance begins when “the final prototype is eventually transformed into the final product” (Hekmatpour & Ince, 1988). This point of view differs considerably from that of the generational framework.

Prototyping has also been equated to implementation techniques such as fourth generation languages. This type of prototyping is of use in the elaboration stage during any generation since we consider any means of realization to be acceptable. We believe that the generational framework is superior to prototyping, regardless of the interpretation.

The spiral model of software development and enhancement (Boehm, 1988) realizes some of the features of the generational framework. In a sense, a spiral is formed by stringing a number of generations of the process of the generational framework end to end. The spiral model also assumes the evolutionary viewpoint of system development; each cycle of the spiral can be substantially different from other cycles; and each cycle of the spiral can be tailored to some more traditional process model. However, the spiral model does not distinguish the four stages of the process nor does it take care of the object at all. This latter problem makes it difficult for the spiral model, in particular, to handle the three cases of system evolution described in the next subsection.

Unlike the generational framework that emphasizes mutual understanding and agreement between customer and developer during the process of system development, the spiral model focuses mainly on the developer side and uses a

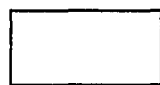
risk-driven approach to control the development process. That is, the criterion of entering a new cycle of the spiral is based on the amount of risk the developer has to take. Since the generational framework does not prescribe how customer and developer decide to spawn new generations of the process, we suppose that the concept of risk management can be borrowed from the spiral model as a way of reaching such a decision.

Arguments similar to those applied to prototyping and the spiral model show the generational approach to be of greater utility than most other existing models. Those models are found wanting in that some do not model the object at all, some do not recognize the relationship between the process and the object, some do not encourage customer/developer interaction, and some do not take the evolutionary nature of systems into consideration.

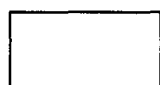
2.3.3 An Application of the Generational Framework

Orthogonal to the types of maintenance cited in Section 2.2.1, a classification of the evolution of systems might include: extending an existing system to a new generation, merging a number of existing systems into an integrated one, or splitting an existing system into a number of derived ones. The generational framework can be used to model all of these cases.

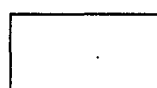
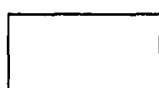
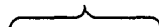
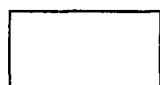
Figure 2.5(a) depicts the basic case in which a new generation of a product is spawned. Figure 2.5(b) shows the merging of two products to create a new one and Figure 2.5(c) shows the division of one product into two new ones. In case (b), a new system is created by merging the objects of two previous systems. In case (c), two new systems inherit their objects from a previous one. The three cases can be cascaded.



(a)



(b)



(c)

Figure 2.5: Evolution of systems.

2.4 Concluding Remarks

We have proposed in this chapter a generational framework that models the information system development process and the object to be developed in a more natural way. This framework

- emphasizes the relationship between customer and developer by identifying a balanced four-stage process, in which customer and developer interact in consumer/supplier manner;
- incorporates the object worked on cooperatively by customer and developer during the four-stage process where the problem part is created during the elicitation and specification stages and the solution part is added during the elaboration and animation stages; and
- features the generational nature of the process and the object, which facilitates the evolution of systems.

This generational framework lays the foundation for the investigation of the requirements determination problem. In the terms defined in this chapter, requirements determination is concerned with the problem representation process and the problem structure of the object. The individual dimension and the group dimension of requirements determination are discussed in the next two chapters.

CHAPTER 3

SYSTEM REQUIREMENTS DETERMINATION

The purpose of this chapter is to investigate the way information system developers perform in the area of requirements determination. We first define the scope of the requirements determination work on the foundation of the generational framework laid in the previous chapter. We then study how system developers conduct the process of requirements determination and how system developers record the requirements. Study of the elicitation process and specification representation helps infer the information needs of system developers in later chapters.

3.1 Requirements Determination Defined

We have proposed in the previous chapter that the information system development process consists of the tasks of elicitation, specification, elaboration, and animation. Elicitation is concerned with acquiring all the information related to the required system. Specification is concerned with formalizing and specifying in a clear and consistent manner the descriptions of the system that fulfills the requirements. Here we adopt the term *requirements determination* to mean the joint process of elicitation and specification, that is, the upper part of a generation of the development process shown in Figure 2.1. Requirements determination is therefore concerned with building up the problem structure of the object being developed (Figure 2.2).

Generally speaking, the process of information system development is one

that involves creating some artifact at almost every stage during its progress. Requirements determination, as part of the system development process, inherits this nature. The process of determining requirements has been observed, in a general context, to consist of two parts: refining a simple goal statement into a set of more explicit and measurable items and generating the specifications of the solution that fulfills those required items (Thomas & Carroll, 1979, 1981). This means that elicitation and specification essentially form a microcosm of the generational process.

Based on this understanding, we propose that the work of requirements determination forms a lower level generational process. Elicitation is a projection of the first two stages of the macrocosmic generational framework and specification is that of the last two stages. Accordingly, the macrocosmic problem structure is decomposed into a problem structure and a solution structure of the second level. The second-level problem structure contains the descriptions of user needs and constraints (see Section 2.2.2) and is the major concern of the elicitation phase. The second-level solution structure contains the descriptions of system behavior (see Section 2.2.2) and is the major concern of the specification phase. Figure 3.1 illustrates the concept of problem and solution structures at both microcosmic and macrocosmic levels.

The distinction between problem and solution structures is that of *given* versus *generated*. This applies to both microcosmic and macrocosmic levels. At the microcosmic level, user needs and constraints constitute the problem structure and represent the given, while the solution structure contains the generated descriptions of system behaviors. At the macrocosmic level, all the descriptions in the problem structure constitute the given, which include user needs, constraints, and system behaviors. The solution structure is generated from this given.

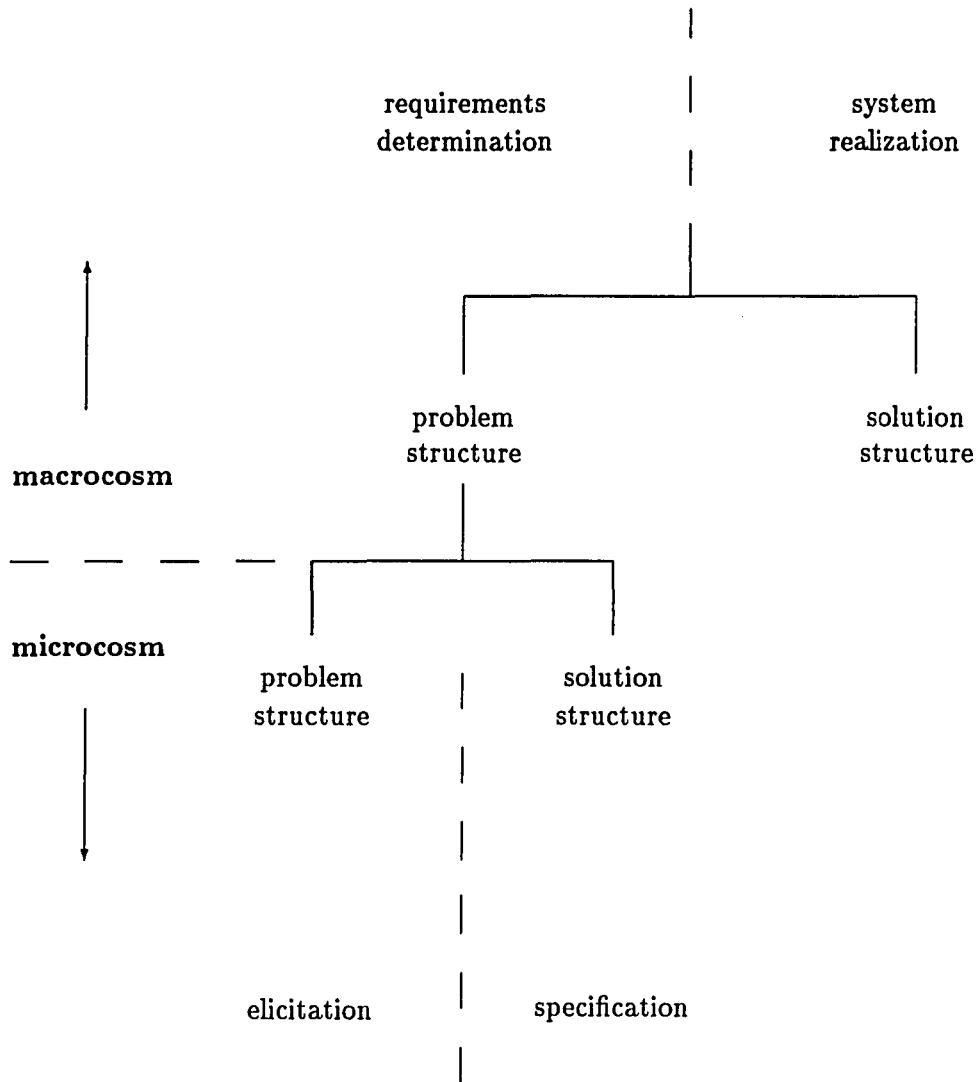


Figure 3.1: The problem and solution structures for requirements determination.

3.2 Requirements Elicitation

3.2.1 Problem-solving Process

Some authors have conjectured about how system requirements *should* be obtained (for example, Davis, 1982; Yadav, 1983). Many more (for example, those reviewed by Connor, 1985) have recommended methods for recording system descriptions and maintain that the process of deriving requirements should be guided by the representations. In other words, those previously proposed techniques focus on the outcomes of requirements determination but not on the process via which the outcomes are achieved. Our interest is in the way developers conduct the task of requirements determination in order to obtain system requirements; that is, the nature of the requirements elicitation process itself.

Among the limited literature on the behaviors of information system developers, most focus on comparing expert and novice programmers in simple program coding or debugging assignments. Also, some researchers identify their studies in the area of “programming,” which covers part of requirements determination, part of system design, part of program/algorithm design, and part of coding (for example, Koubek, Salvendy, Dunsmore, & LeBold, 1989). This is mainly because the researchers do not have the orderly concept of system development in mind. In short, when comparing previous works, we have had to be very careful in sieving the information which is really relevant to requirements determination. To determine whether a work is relevant, we give the greatest weight to the problem assignments given to subjects during the research.

Vitalari and Dickson (1983) identified their work in the area of “system analysis,” which is the conventional term of requirements determination. They observed the characteristics of the requirements determination problem and postu-

lated that the problem is cognitive in nature and that a problem-solving approach should be used. The assignment given to the subjects in their experiment was about developing the problem structure and part of the solution structure at the microcosmic level. They concluded from the experiment an account of highly-rated system analysts' problem-solving behaviors:

- A highly-rated analyst collects information from the environment to classify the problem and relate it to previous experience. If a match is found, the analyst draws upon that previous problem experience to practically structure the current problem, search for additional information, and in some cases employ previous solutions.
- Highly rated analysts use a hierarchy of measurable goals to structure the overall task. With this, they can deal with the problem at different levels of detail.
- A highly rated analyst is adept at managing hypotheses in the problem-solving process by discarding low probability hypotheses and retaining valid ones.
- Highly rated analysts understand the importance of the interpersonal relationship between the analyst and the user and consequently allocate time to maintain a productive relationship with the user.

In many cases, the word "design" has been used to denote all the works prior to programming, although different authors may not agree on the scope covered by design. Malhotra, Thomas, Carroll, and Miller (1980) studied design in general contexts. One of their experiments required subjects to write functional requirements (microcosmic solution structure) from a set of given information (microcosmic problem structure). The finding was that the results generated by all

subjects were widely divergent.

Adelson and Soloway (1985, 1988) used the term “design” to include requirements determination, system design, and to some degree, program design. Three assignments were used in their studies. Although Adelson and Soloway called the assignments “design specifications,” they were in fact the microcosmic level problem structure. The subjects were expected to generate the solution structures of both the microcosmic and the macrocosmic levels, where the latter was expected to be as detailed as pseudocodes. From their studies, Adelson and Soloway have identified these problem-solving behaviors of designers:

- forming a mental model,
- systematically expanding the mental model,
- mentally executing the mental model,
- making notes,
- representing constraints, and
- labeling and retrieving models.

When designers are given a problem, they first form mental models of how they perceive the problem. A mental model is usually something that helps decompose the problem into smaller ones and helps organize those small parts together. Designers expand their mental models into more and more details as time elapses. From time to time, designers execute mental models to check whether they really represent solutions to the problem. Note making helps designers expand mental models systematically. When the designers are not familiar with a (sub-)problem, they represent constraints explicitly to clarify it and subsequently decompose it. When they encounter a familiar (sub-)problem, they mark it with the label of a retrievable model.

In the study by Guindon, Krasner, and Curtis (1987), the term “design” was equated to system design and algorithm design. Given a more difficult and detailed (as compared with other studies) statement of functional requirements that fit in the macrocosmic problem structure, subjects were required to develop part of the macrocosmic solution structure that could be handed off to a programmer to implement. This study is peculiar in that it does not propose what designers do but identifies the *breakdowns* that could happen when designers do their work. Although the scope of this study (system design and algorithm design) deviates from our interest, the concept of breakdowns crosschecks the behaviors identified in Vitalari and Dickson’s and Adelson and Soloway’s studies.

All these previous works that are related to our interest assume a *design* problem-solving, instead of *general* problem-solving, approach. General problem-solving presumes the existence of a fixed goal state and a fixed space of stages with respect to given constraints. The problem-solving process involves traversing the space to achieve the goal state. The solution usually has no variants when the goal and constraints are given. That is, the solution comes as a natural result of elaborating the goal and constraints, but not inventing something new. Design problem-solving is different in that it does not presume the goal or the space to be fixed. Given the goal and constraints, the design problem-solving process is an exploratory process with which developers could generate a number of alternatives satisfying the goal and constraints. Among the alternatives, one has to be designated the solution.

The concept of design problem-solving is reflected in the two-level problem and solution structure proposed in the previous section. An information system is considered to be an artifact. An information system developer has to elaborate the goals and constraints in the microcosmic problem structure and generate and

select a set of descriptions of system behavior in the microcosmic solution structure. This process happens again when the developer takes the macrocosmic problem structure as the given to generate the macrocosmic solution structure.

Another commonality among the above-mentioned studies is that all of them studied only new development. Hunt (1987) argues that any particular design project is likely to be a mixture of new design and redesign, where “redesign is typified by explicit and purposeful acceptance of much or most of an existing specification with the intent of changing some portion to make it more effective, efficient, safe, or less expensive to use or manufacture” (p. 146). With the generational view of information system development discussed in the previous chapter, we regard the development process of subsequent generations as being isomorphic to the original development process. This consideration surely eliminates the argument between new development and re-development. We therefore consider that the results from those previous studies apply equally to new development and re-development.

3.2.2 Engineering Design Process

Engineering design has been considered a course of action for the development of an artifact where the process involves transformation of an initial tentative description of the artifact into a more or less complete one (Gorb & Dumas, 1987; Lansdown, 1987). The characteristic of developing artifacts in engineering design coincides with the characteristic of developing artifacts in information system development. We therefore believe that we can learn about information system development by comparing it with engineering design.

In the engineering field, the word “design” has been used to denote a wide spectrum of activities, for which different authors use diverse terminology. For one

example, the design activities may include conceptual system design, preliminary system design, and detail design, and each may consist of a series of subordinate activities (Blanchard & Fabrycky, 1990). Comparing this to information system development, the scope of engineering design is roughly equivalent to the combination of requirements determination, system design, and detailed design. Yet, authors in engineering design rarely distinguish requirements determination work from others. The whole design process is usually treated as an entity. When we are trying to borrow from the literature of engineering design, we also have to be very careful in selecting the information relevant to our topic of interest.

Since empirical data about engineering design are sparse, authors tend to present engineering design process in the format of itemized lists instead of a coherent description of the process. John (1988) conducted a series of field studies of unstructured interviews and identified a list of attributes of engineering design. Those relevant to our discussion are:

- Design is a process of clarifying goals and constraints iteratively, commencing with incomplete and unstructured problems and concluding with solution formulations that clarify goals.
- Design does not necessarily involve optimizing solutions because of the constraints.
- Design often reflects the needs, values, and purposes of designers in orders and patterns that give meaning.
- Design is a multi-disciplinary activity demanding communication boundaries.

Considering the problem and solution structures proposed in the previous section, we can infer from these attributes a scenario of design process. A designer conducts the process by gradually expanding the problem structure and

the solution structure. In the case of requirements determination, the problem structure contains the goals and constraints and the solution structure contains a set of system specifications that satisfy the goals and constraints. During the course of design, designers may have to make judgments that reflect their styles and experiences. They may also consult a number of information sources.

The idea of regarding design as expanding the problem structure and solution structure is well supported by the results from a series of experiments concerning how architects design. Akin (1978) identified two major phases in the architectural design process: pre-sketching phase and sketching phase. The former is about acquiring information in order to interpret and represent the problem and is essentially about expanding the problem structure. The latter is about generating, integrating, and assessing solutions and is essentially about expanding the solution structure.

Meister (1987) contends that the elements in the design process, in addition to those attributes proposed by John, include building a mental model of the system being developed. The mental model is progressively constructed during the course of design and contains all information accumulated along the progress. The mental model serves as a sort of outline that helps designers review at each stage what they have accomplished so far and what they must do to work out their assignment.

An often cited problem regarding the process of engineering design is that of top-down (analytic) versus bottom-up (artistic) (Rouse & Boff, 1987). It is commonly considered that system "objectives are usually stated in terms of system performance capabilities which may then be decomposed into functional requirements" (Rouse & Cody, 1988, pp. 230-231). That is, the system requirements are represented in hierarchical manner. The argument is whether the development

process should be top-down decomposition as the representation is or the process should be bottom-up composition, acknowledging that each approach has pros and cons. Rouse and Boff summarize a workshop participants' opinions that the development process involves a mixture of top-down decomposition and bottom-up composition approaches, with more emphasis on decomposition earlier in the process and a shift toward composition later in the process.

Under the problem and solution structure framework proposed in the previous section, system objectives and performance capabilities are what we have called goals and constraints and should go into the microcosmic problem structure. The functional requirements (system behaviors) then belong to the microcosmic solution structure. Consider the difference of *given* and *generated* between the problem and solution structures. When developers are acquiring the information for the microcosmic problem structure, they should be able to do it analytically in a top-down manner as the goals and constraints are given. When developers are generating the system behaviors for the microcosmic solution structure, they may have to do it in bottom-up manner at least from time to time as this artistic approach fosters creativity and innovation. Our interpretation supports this opinion about the mixture of top-down and bottom-up approaches.

3.2.3 Requirements Elicitation Process

We have studied in the previous two subsections the empirical studies in both the information system development field and the engineering design field. From the limited and diverse literature, we have tried to interpret those results within the scope of our framework, which was presented in Section 3.1.

It has been admitted that the few process models proposed in both the information system development and the engineering design fields are speculative and

not provable (Guindon & Curtis, 1988; Meister, 1987). It also has been observed that research into the programming-in-the-large area (which roughly includes requirements determination and system design) is in a mode of theory formation, instead of theory testing (Soloway, 1986). With these caveats in mind, our intention is to cross-examine the findings from both fields and extract the most useful information.

Based on the information from the previous two subsections and on our judgment we present the elements of requirements determination in Figure 3.2. There are two major elements that interact: the mental model and the set of design problem-solving behaviors. The mental model basically contains the goals and constraints of the system under investigation, the functional requirements of the system that fulfill the goals and constraints, and some of the developer's knowledge that is relevant to the requirements determination assignment. The goals and constraints constitute what we have called the problem structure and the functional requirements constitute the solution structure, both of the microcosmic level. The structures of both will be discussed in the next section. The developer's knowledge contains at least some models that could be used when the developer is exploring the problem structure or appending the solution structure.

Information collection is a major behavior that takes place during requirements determination, especially requirements elicitation. The information collected by the developer from the customers and their environments goes mainly into the problem structure to set up the goals and constraints of the system under study. To the developer, interpersonal communication is the behavior that facilitates information collection. For a project that has multiple developers and/or has the customers actively participating in the development process, communications among all the participants become much more complex and important. This will

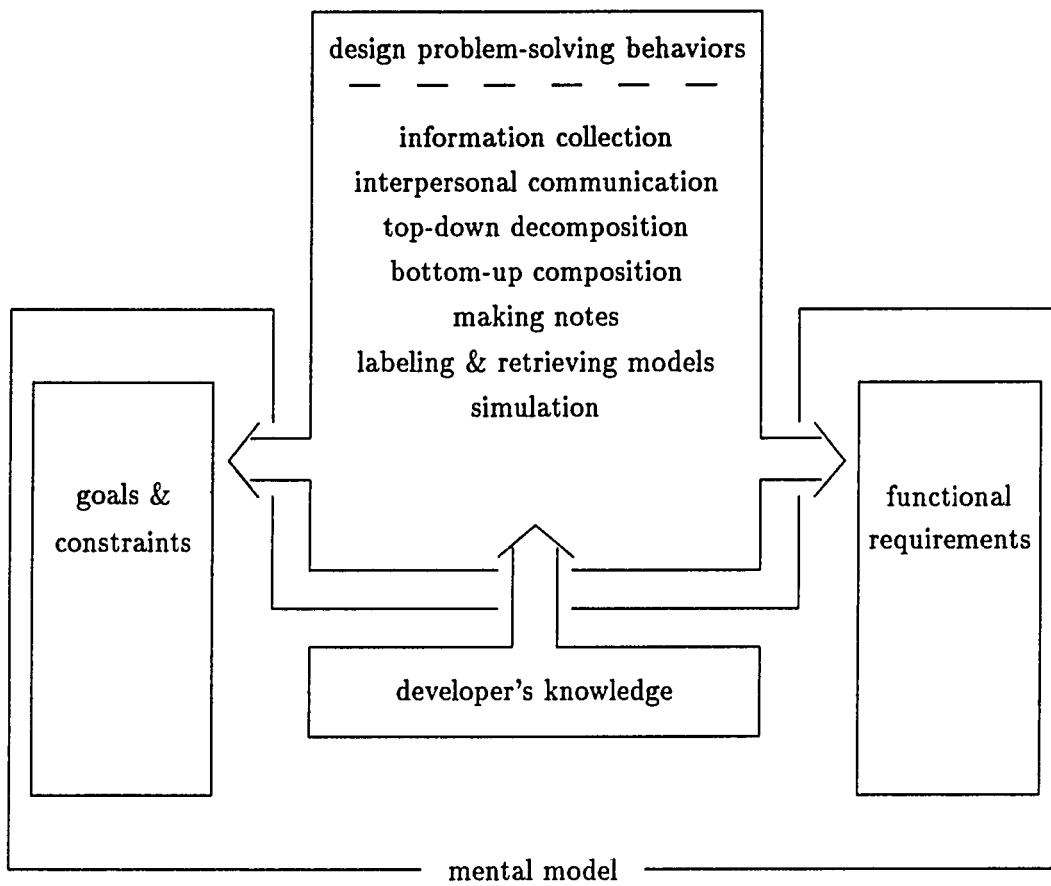


Figure 3.2: The elements of requirements determination.

be discussed in the next chapter.

The expansion of the problem structure and the solution structure is mainly a mixture of top-down decomposition and bottom-up composition. Though it might depend on the developer's style and preference, top-down decomposition happens mainly in exploring the problem structure and bottom-up composition is mainly used in generating the solution structure. From time to time, the developer may switch from top-down to bottom-up and vice versa. This interplay of top-down and bottom-up behaviors has been called a *heterarchical* process (Rouse, 1986).

During the heterarchical process of requirements determination, the developer may make a note about an item in the problem structure and solution structure, indicating that he has to divert attention to another topic but needs to come back to the original item. Whenever the developer recognizes that a certain item matches what already exists in the models he possesses, he may label that item and not explore it any further. Occasionally, the developer has to check the integrity of the problem structure or the solution structure or to evaluate the correctness of the correspondence between those two structures. Simulation of the problem and solution structures helps test their integrity and correctness.

We have proposed that requirements determination process is a microcosm of the generational process of information system development. This means that the developer will switch back and forth between the problem structure and the solution structure. This happens especially after a simulation concerning the solution structure is done. From results of the simulation, the developer may find inconsistency between the two structures and has to go back to the problem structure to remedy the problem and then proceed to regenerate the corresponding parts in the solution structure.

3.3 Requirements Specification

3.3.1 Requirements Specification Problem

The emphasis of requirements specification should be on the organization of the macrocosmic problem structure that is manipulated during the requirements determination process. Some authors (for example, Rowen, 1990) focus the study of requirements specification on the formats of the documents to be produced. We consider this approach inappropriate. Documentation formats are ways of presentation. A documentation format does not necessarily reflect all views of the intrinsic organization of the whole problem structure. Studying presentation formats rather than the intrinsic organization inhibits our understanding of the real problem and, therefore, does not serve our interest.

Our interest is in the intrinsic organization of the macrocosmic problem structure which developers can make the most of during requirements determination. Considering that the macrocosmic problem structure consists of a problem structure and a solution structure of the microcosmic level,¹ the problem of requirements specification has three parts:

- the organization of the problem structure,
- the organization of the solution structure, and
- the interrelationship between the two structures.

The basic components of the intrinsic organization of both the problem and the solution structures are objects (entities) and associations (relations), where the latter are the links between the former (Dubois, Hagelstein, & Rifaut, 1988;

¹Hereafter, when we mention problem structure and solution structure, we mean the microcosmic-level ones, unless otherwise specified in the context.

Carroll, Thomas, Miler, & Friedman, 1980). The discussion in the remainder of this section concerns the objects and associations that constitute the three parts of the requirements specification problem.

3.3.2 Problem Structure

Hierarchical structures have traditionally been used in describing systems. For one example, Simon (1962) argued that virtually all complex systems, from social to biological to physical to symbolic, are hierarchical in nature. For another example, by showing cases of grouping requirements into hierarchically related sets of classes, Alexander (1964) suggested that the representation of design problems should be hierarchical. Alexander's idea has since been considered a conceptual basis for the evolution of structured techniques, including step-wise refinement to program development and top-down decomposition to system design. The concept of hierarchy has also been found to be useful in the realm of system requirements analysis.

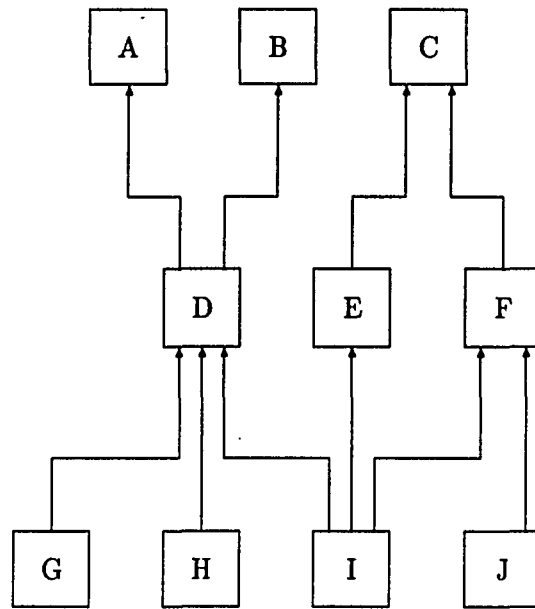
Davis and Olson (1985) propose that an information system should be described in terms of a hierarchy of *goals*, *objectives*, and *strategies*. Similarly, Shamlin (1989) contends that the specifications for the requirements of a system contain a related set of *needs*, *goals*, and *objectives*. The needs are the basics of the system requirements. Each need is translated into a number of goals that specify the ways to achieve the need, where one goal may address multiple needs. Each goal is in turn addressed by a number of measurable objectives. However, there are difficulties common to approaches like these two.

The first difficulty is the ambiguity of terminology. Terms like goal and objective are not usually immediately distinguishable by their definitions. As an example, Kinston (1986) collected from the literature more than 30 synonyms

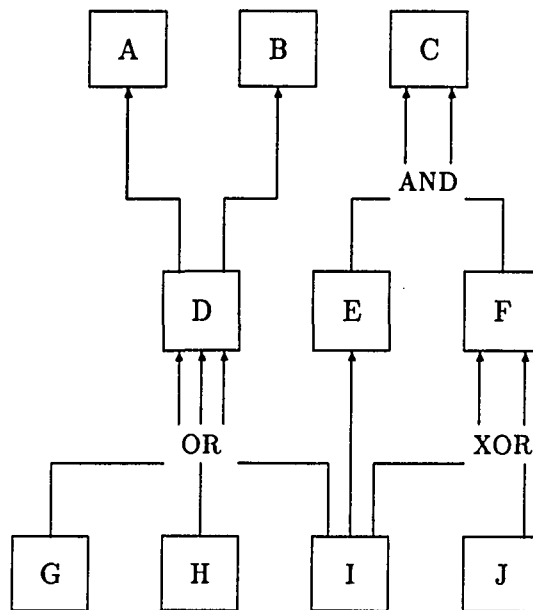
(each having its nuance) of goal or objective. What is worse, different authors often use different terms to label the same level of the decomposition or use the same term in different levels. The second difficulty is the fixed number of levels of decomposition. We believe that an appropriate representation should be used in each different situation. Whether system requirements can be specified in only two levels or in as many as seven levels, the information should be captured as it is. The third difficulty is that the requirements “hierarchy” is not a pure hierarchy. For example, a goal may support more than one need in Shamlin’s scheme.

The *intent structure* (Warfield, 1973) is a way of representing the problem structure while avoiding the three difficulties. An example intent structure is shown in Figure 3.3(a). The only object type used in the intent structure is *objective*, which is a boxed item, and the only association type is *supports*, which is a link between two boxes. For example, objective D supports objective A, and objective C is supported by objectives E and F. The first difficulty is avoided by using one term throughout all levels of the intent structure. The choice of the term objective is arbitrary. Any synonym applies equally well. The second difficulty is solved by not limiting the number of levels. The third difficulty is handled by explicitly permitting a lower-level objective to support more than one upper-level objectives. This results in so-called *multiple-channel hierarchies* (Nadler, 1981). Furthermore, as shown in Figure 3.3(b), logical connectives like *and*, *or*, and *exclusive-or* (XOR) may be inserted between levels of objectives to enhance expressiveness.

Each objective in an intent structure is a statement of the form “To ‘Verb’ ‘Object’ ‘Qualifier’.” For example, an objective might read “To ‘provide’ ‘sales forecasts’ ‘that are accurate’.” This standard form can be reduced to an unqualified form of “To ‘Verb’ ‘Object’” and a simple form of “‘Object’” (Volkema, 1988). Examples are “To ‘provide’ ‘sales forecasts’” and “‘Sales forecasts’.” When the



(a)



(b)

Figure 3.3: Example intent structures.

standard form is transformed to the unqualified and the simple forms, the meanings become broader. If we put the above three examples into an intent structure, we should see that the simple form example is in a highest level while the standard form example in a lowest level. We regard this Verb-Object-Qualifier objective format as being very useful in describing system needs.

Referring back to Section 3.1, the problem structure contains not only needs or goals but also constraints. We observe that the Verb-Object-Qualifier objective format can also be used to express constraints. The key is that a goal and a constraint are the two sides of a coin. For example, “To ‘provide’ ‘sales forecasts’ ‘that are accurate’” could be a goal or a constraint. It is up to customers and/or developers to decide what are goals and what are constraints. With the use of intent structure, the elegance is that goals and constraints can be organized in the same way. In case it is necessary to explicitly distinguish goals and constraints, items in an intent structure may be so annotated.

3.3.3 Solution Structure

Researchers have proposed many representation methods (see, for example, those reviewed by Davis, 1988a, and Webster, 1988) for describing system specifications, which are the concern of the solution structure. Yet, no one specific method has been considered universal. In order to avoid the problem of sticking with one or a fixed set of incomplete representation methods, we espouse the idea of the *federation approach* (or *metasystem approach*) of using a single scheme to support multiple representations (Loucopoulos, Black, Sutcliffe, & Layzell, 1987; Kottemann & Konsynski, 1984).

The multiple-channel hierarchy for the problem structure is in fact a miniature of the federation approach where one object type and one association type

are involved. The *objective* object type and the *supports* association type that constitute the “language” for the problem structure are derived from the generic concept of object and association. By the same token, the “language” for the solution structure is obtained by deriving an appropriate set of object types and association types. Examples of the object and association types for use in the solution structure can be found in Dubois et al. (1988) and Loucopoulos et al. (1987).

The last part of the requirements specification problem is the interrelationship between the problem structure and the solution structure, that is, which parts in the problem structure are fulfilled by which parts in the solution structure, and vice versa. Since both the problem and the solution structures are governed by the concept of object and association, the interrelationship between the two structures can be achieved by an association type that connects the object types in the problem structure and the counterpart object types in the solution structure.

3.4 Concluding Remarks

Continuing the discussion in the previous chapter about the overall process of information system development, we have focused the discussion in this chapter on requirements determination and the subordinate tasks of requirements elicitation and requirements specification. The process of requirements determination is a projection of the generational process of information system development. In the latter case, a set of macrocosmic problem and solution structures are involved; in the former case, a set of microcosmic problem and solution structures are involved (see Figure 3.1).

The process of requirements elicitation, as we have examined the view-

points from both the information system and the engineering design fields, is a design problem-solving process involving a number of problem-solving behaviors. These behaviors are applied on a mental model (Figure 3.2), which consists of the developer's knowledge and the problem and solution structures. The problem structure contains the goals and constraints of the system under development and the solution structure contains the specifications of a system fulfilling the goals and constraints. We have suggested a multiple-channel hierarchy (Figure 3.3) for recording the goals and constraints in a unified manner and a federation approach for recording system specifications.

The discussions in this chapter are mainly about how individual developers conduct the requirements determination process and what representation structures best support the process. In the next chapter we will discuss how a group of people participate in requirements determination and how they can benefit from the process and structures we have concluded in this chapter.

CHAPTER 4

REQUIREMENTS DETERMINATION BY GROUP

We have discussed in the previous chapter how system developers perform the requirements determination process and what representation structures best support the process. On top of these results, we want to study in this chapter how a group of developers, customers, users, and other interested parties can participate in the requirements determination task. We first investigate different degrees of participation and name the one that is suitable to our research area, and then present a description of the process of group approach to requirements determination with respect to the representation structures proposed in the previous chapter. We also consider ways of selecting people to participate in the requirements determination task.

4.1 The Problem of Participation

Many authors have discussed from the standpoints of technical people the many advantages of involving non-technical people in the process of system development. For example, Nadler (1981) postulates that the involvement of people “can maximize the number and effectiveness of implemented solutions and the effectiveness of utilizing ... resources” (p. 207). For another example, Ward (1987) thinks that group processes “are more powerful than individual processes [and] ... are capable of rendering deeper, more profound solutions” (p. 157).

This idea of involving people has been reflected in the evolution of engineer-

ing design methods, as summarized by John (1988). The first-generation method is called systematic design, which assumes that technical people know what users need. The second-generation method is called participative design. This method assumes that users know what they need and can direct the progress of the design. The third-generation method is called interactive design, where the technical and non-technical people share responsibilities. Following a similar approach, we have established a five-category classification of the degree of participation in the development of information systems.

The first category is presumably analyst-dominated and the user has minimal involvement. In what Ginzberg (1979) called "conventional" approach, for example, analysts are active and users are passive during system development. After being given the assignment of developing a system, the analyst examines the current system and prescribes a new system. The user's only role is to respond to the analyst's questions, if any, and to use the new system. Results from White and Leifer's (1987) questionnaire survey show that a large portion of technical people take this position and do not think the involvement of users important.

The second category, as an improvement to the first category, is for the technical side to recognize the user side. This approach is exemplified by King and Cleland's (1975) method. They contend that in order for an information system to be usable, the user should be involved in the development. What users can do, in addition to responding to analysts' questions regarding the system to be developed, is to "sign off" on what analysts have done. That is, the development is mainly performed by analysts but users have rights to agree or disagree. Results from Kaiser and King's (1982) survey show that this type of participation has been prevalent.

The third category extends the involvement from the non-technical side to

multiple parties. In this situation, as described by Zmud (1980), "representatives of the functional area(s) sponsoring a project should compose a majority of the participants, and the leader of the requirements analysis effort should be the functional manager who is immediately responsible for the sponsoring organization" (p. 51). Technical people are not eliminated; they participate on an as-needed basis. Those successful cases reported by Kozar and Mahlum (1987) and Leonard-Barton (1987) adopted this user participation policy. The team organization suggested in Joint Application Design (IBM Information Services, 1987) also reflects this kind of involvement.

The fourth category, parallel to the third category, is extending the involvement to multiple parties on the technical side. In their experiments, Malhotra et al. (1980) observed that design specifications are often incorrect and incomplete because different designers paying more or less attention to different aspects of the design. This observation provides for a motivation of involving multiple specialists in system development. Distaso's (1980) suggestion of the "team approach" that involves multiple system specialists is an early form of this category. The project meetings recorded and analyzed by Walz, Elam, Krasner, and Curtis (1987) are also of this category.

The fifth category is an extension of the third and the fourth categories. This kind of participation involves multiple parties from both the technical side and the non-technical side. There are two natural consequences of this kind of participation. The first is the use of *consensus approach* for conducting the process of development.¹ Multiple parties from both the technical and non-technical sides

¹Mumford (1979) uses the term consensus differently to mean the participation of *all* people concerned, for example, all workers whom could be affected by an information system to be installed at a work place.

have equal rights and share the project. Decisions are made by all participants on a consensus basis. As observed by Ginzberg (1979), the consensus approach has the effect of blurring the distinction between analysts and users because each individual is recognized as a specialist whose contribution is necessary to the development process.

The consensus approach to system development has been adopted before. For example, both the projects reported by Leonard-Barton (1987) and Walz et al. (1987) emphasized team consensus. However, this approach has not been well realized because the biased team organizations of the first four categories do not naturally facilitate it. We believe that the team organization of this category of participation can make the consensus approach to system development possible.

The second consequence of the shared team organization is that the role of an analyst becomes a role of facilitation or support (Srinivasan & Davis, 1987). Since all participants share the responsibility of system development, the analyst no longer has to take the whole responsibility. In other words, because of the consensus approach the analyst can no longer dominate the process of development.

It is our interest to study the use of the last category of participation in requirements determination, which we call the group approach to requirements determination. The problem of the group approach to requirements determination has two parts: how a group of participants work together to derive the requirements and who to participate in requirements determination. These two parts are discussed in the next two sections, respectively.

4.2 The Group Approach to Requirements Determination

In order for the group approach to requirements determination to materialize, some conditions have to be met (Wulz, 1986):

1. All persons involved are known as individuals.
2. These individuals are interested in and motivated for participation.
3. These participants have time to be involved throughout the project.

We assume that these conditions hold.

It was suggested in the previous chapter that a developer holds a mental model during requirements determination. This mental model consists of the developer's knowledge as well as a problem structure and a solution structure (see Figure 3.2). Now that every person participating in the group approach to requirements determination is regarded as a specialist sharing the responsibilities of the task, each participant, whether from the technical side or the non-technical side, holds a mental model describing what this participant thinks of the system under study. The process of requirements determination is then a process of seeking a consensus model that accommodates most of the individual mental models.

The knowledge pool of the consensus model is formed by concatenating the knowledge in individuals' models. This means that the knowledge available for use by a group is the composition of the knowledge possessed by all participants. This large pool of knowledge facilitates the progress of requirements determination by providing for more information that could be used in elaborating the problem and solution structures. We will discuss in the next section the issue of forming groups that possess adequate knowledge pools.

The problem structure and solution structure of the consensus model are obtained by incorporating the similarities and resolving the differences of the problem

and solution structures in the individual models. This practice has been successfully tested in the architectural design field (Ward, 1987). Each of the architectural students participating in a studio built a scale model of the design assignment. Photographs of the models were taken from the same point of view. These photographs were then projected onto one photograph by multiple-exposure technique. On the multiple-exposure photograph, the similarities of the participants' designs clearly showed up. The differences among the participants' designs could also be identified. This process could continue until all discrepancies are resolved. That is when a coherent group design is achieved.

The essence of the group process of building the consensus problem structure and solution structure is searching for as many similarities as possible among the structures held by the participants and trying to resolve any differences that emerge gradually through group efforts. While deriving a model which has the greatest commonality is the end, determining the discrepancies is the means. The group approach to requirements determination is therefore a process of dealing with *issues*, where the issues are the discrepancies among the participants' models. In the meanwhile, the act of solving open issues is a compound of group efforts and individual efforts. Participants may individually perform the behaviors we collected in the pervious chapter (see Figure 3.2); the whole group may also perform those behaviors together. The participants revise their individual mental models whenever an issue is solved.

While a group is dealing with unsolved issues, new issues will usually surface and will eventually have to be handled. That is, new issues make the process ongoing. The group process of reconciling issues continues until there is no pending issue or the group decides not to consider new issues or new information. This iterative process reflects what we have asserted in the previous chapter that the

process of requirements determination is a projection of the generational process of information system development.

4.3 Composition of Group

As discussed in the previous section, we want to accumulate a pool of knowledge from the group participants. The objective is that the participants possess diverse characteristics and that the overlap of the characteristics they possess is minimal. We call groups having this property *heterogeneous*. To facilitate selecting people to form groups, we need to compare the degrees of heterogeneity of groups. The basic idea is to assign a quantitative measure, called group index, to each group. A numerically larger number indicates that a group is more heterogeneous. A numerically smaller number indicates that a group is less heterogeneous. That is, the group index should reflect the relative differences between groups. The formulation is in the following.

Let G be the set of groups under study and C be the set of all characteristics applicable to G . For each $g \in G$, define

$P_i : C \rightarrow \{1, 0\}$, $i = 1, \dots, n$, where n is the number of participants in g

$$P_i(c) = \begin{cases} 1 & \text{if participant } i \text{ in } g \text{ has characteristic } c \\ 0 & \text{otherwise} \end{cases}$$

and

$$F_i = \{c \mid c \in C \text{ and } P_i(c) = 1\}$$

$$D_{i,j} = F_i \oplus F_j$$

Then, the index of group g is

$$H_g = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n |D_{i,j}|}{n(n-1)/2}$$

Our approach to the definition of group index is, first, to calculate the differences between participants; second, to accumulate all the differences; and, third, to “normalize” the total differences over the group. P_i is a function that determines whether the participant i of group g has a characteristic c or not. If yes, its value is 1, otherwise, 0. F_i , which is a set, is the profile of participant i as it contains all the characteristics whose P_i value is 1. $D_{i,j}$ is the exclusive union of F_i and F_j and indicates the differences between the two participants i and j . $|D_{i,j}|$ denotes the size of $D_{i,j}$ and is the numerical value which we believe can denote the differences between profiles of two participants. The group index H_g for group g is thus the average value of all $|D_{i,j}|$.

The group index is defined for $n \geq 2$. The minimum value is 0, when all profiles for g are the same. The maximum value is the size of C , when all profiles for g are different. All $g \in G$ may not have the same number of participants. The index values are valid in that they are averaged over $n(n-1)/2$. However, all $g \in G$ must be categorized according to the same C . When a group g has no participant having certain $c \in C$, c will not appear in any F_i for g and the resultant index values are valid. Most importantly, this group index should only be used as a relative measure, instead of absolute measure. For example, if the index of group A is 3 and that of group B is 7, we can only infer that group B is more heterogeneous than group A.

4.4 Concluding Remarks

We have investigated the degree of participation in the development of information systems and considered that the kind of participation involving both technical and non-technical people in consensus-style process is suitable to require-

ments determination. Based on this understanding, we have proposed how a group of people can work together to derive a consensus model that contains the problem structure and solution structure of the requirements of an information system. We have also defined a quantitative measure for comparing the heterogeneity of groups. This quantitative measure helps in selecting participants to fill in groups that perform the consensus process to derive system requirements.

In this and the previous two chapters, we have focused on the overall process of information system development, on how developers conduct requirements determination, and finally on how a group of people work together in requirements determination. In the following chapters, we will investigate the use of computer aids to facilitate the group endeavor of information system requirements determination.

CHAPTER 5

FITTING COMPUTER AIDS

The purpose of this research has been to fit existing computer aids to the group process of information system requirements determination. To this end, we study in this chapter the general problem of fitting computer aids to task areas. We develop a model for fitting computer aids and, based on the fitting model, investigate the usability and configuration problems of using computer aids in a task area. We divide both problems into a number of subproblems and devise methods for solving each of these subproblems. These methods are then applied in the next chapter to fit a specific set of computer aids to the task area of the group approach to requirements determination.

5.1 The Approach to Fitting Computer Aids

Computer aids are tools for helping users perform certain tasks. Since computer aids are information systems that support the information relevant to the users' tasks, computer aids can be studied in terms of the types of information they can support or in terms of their mechanisms that help generate or acquire the types of information. Therefore, our approach for applying computer aids to the aspects of a task area is to bind two sides together by a classification of information types. This approach leads to the identification of two mappings:

- task aspects \longleftrightarrow information types, and

- information types \longleftrightarrow functionalities or tools of computer aids that provide or help acquire the information.

Examples of these two mappings are shown pictorially in two matrixes in Figure 5.1(a) and (b). Therein, information type I2 supports task aspects T2 and T4 and task aspect T2 needs the support of both information types I2 and I3, functionality F1 of the computer aids supports information types I2 and I3 and information type I2 can be provided by functionality F1 or F2, and so on. The combined mapping between task aspects, information types, and functionalities of the computer aids is shown in Figure 5.1(c) in an annotated graph format.

In fitting computer aids to a task area, we are mainly interested in two related problems: the usability problem and the configuration problem. The usability problem is that of determining whether the computer aids can effectively assist in performing the task. In other words, we want to determine whether the functionalities of the computer aids are enough to support all aspects of the task and whether each functionality of the computer aids can be used in supporting certain aspects of the task. If all task aspects are supported, then the configuration problem is that of selecting a minimum collection of the functionalities of the computer aids for economically supporting all aspects of the task. Based on the foundation of the two mappings presented above, we investigate the natures of the usability and the configuration problems and propose effective methods for dealing with these problems in the next three sections.

The advantage of using two mappings, instead of one mapping directly from task aspects to the functionalities of computer aids, is that we can change either of the two mappings. By changing the first mapping, we can study different task areas. By changing the second mapping, we can study different kinds of computer aids. Another advantage of using two mappings will appear when we describe

Information types	Task aspects			
	T1	T2	T3	T4
I1	1			
I2		1		1
I3	1	1	1	
I4			1	1
I5				

(a)

Computer aids functionalities	Information types				
	I1	I2	I3	I4	I5
F1		1	1		
F2		1		1	
F3			1	1	
F4					1

(b)

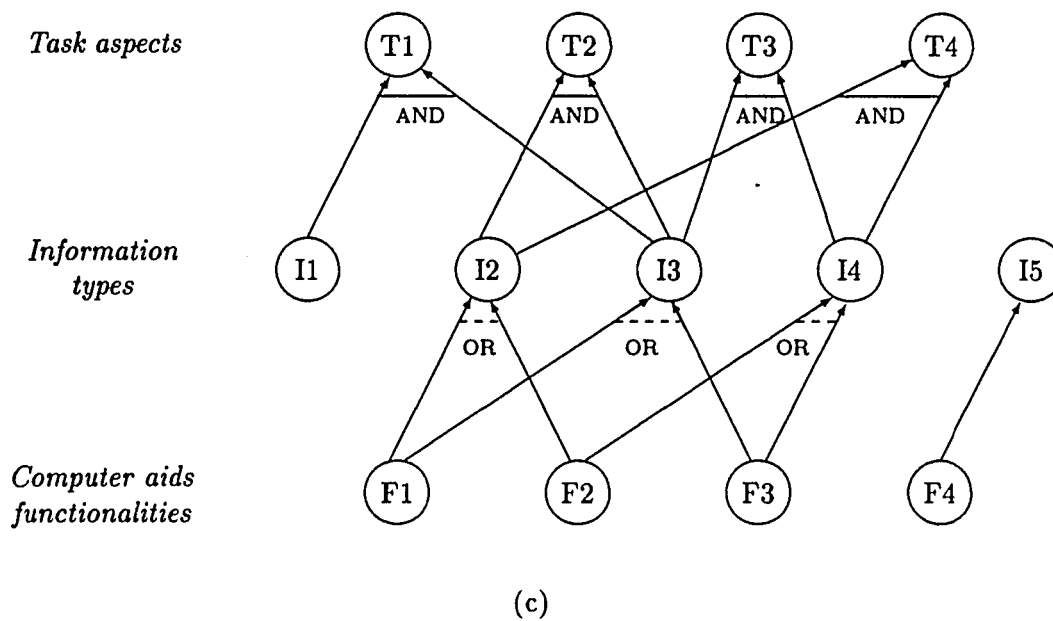


Figure 5.1: Example mappings between task aspects, information types, and computer aids functionalities.

the solution to the configuration problem later in this chapter. In short, the two-mapping model helps decompose the problem.

The two-mapping model and the inferences discussed in the following sections together can be regarded as a means of realizing a variant of the theory on “media selection” (Daft & Lengel, 1986; Daft, Lengel, & Trevino, 1987). Jarvenpaa, Rao, and Huber (1988) made some modifications to the original formulation and tried to match the type of task and type of communication media by characterizing both and finding the fit between them. Our purpose is to match the type of task and the type of computer aids. We characterize the task by characterizing its information requirements and characterize the computer aids by characterizing the information they can provide. The matching then involves comparing the information characteristics of both sides. Essentially, our approach is a realization of Jarvenpaa and colleagues’ formulation of the “media selection” theory.

5.2 The Usability Problem

The usability of a set of computer aids to a task area concerns whether the former can be used to help in performing the latter. Considering the two-mapping model of task aspects, information types, and functionalities of computer aids, we can reformulate the usability problem into four questions:

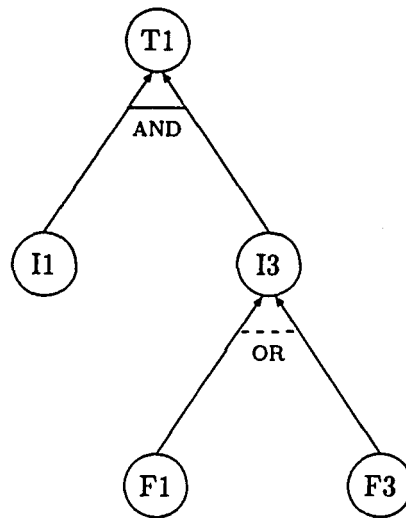
- Is there any information type that is *not* supported by any functionality of the computer aids?
- Is there any task aspect that is *not* supported by the functionalities of the computer aids?
- Is there any functionality of the computer aids that does *not* support any information type?

- Is there any functionality of the computer aids that is *not* used by any task aspect?

Answers to these questions indicate the usability of this particular set of computer aids to this particular task area. Following are the methods for deriving the answers. Those methods are illustrated using the example mappings given in Figure 5.1.

To answer the first question—if an information type is not supported by any functionality of the computer aids, we need to check the lower part of Figure 5.1(c) to see if there are any links leading from any nodes of the functionalities of the computer aids to the node of that information type. This inquiry can be answered by examining the matrix of information types by functionalities of the computer aids in Figure 5.1(b). That the column of an information type does not contain any marked entry means that the information type is not supported by any functionality of the computer aids.

The second question, if any task aspect is not supported by the functionalities of the computer aids, is equivalent to asking if every task aspect is supported by the functionalities. This inquiry has to be done for each task aspect individually. For each task aspect, we need to figure out its connections with information types and in turn with functionalities. As an example, a subgraph that is relevant to task T1 in Figure 5.1(c) is reproduced in Figure 5.2(a). A matrix equivalent to this subgraph is in Figure 5.2(b). This matrix is obtained by first extracting the T1 column of the task aspects by information types matrix (where I1 and I3 are marked) then using this vector to “mask” the columns of the matrix of information types by functionalities of the computer aids (where the I1 and I3 columns are picked). As in the first inquiry, we can examine the matrix instead of tracing the subgraph. Any empty column indicates that the task aspect requires the support



(a)

Computer aids functionalities	Infor- mation types	
	I1	I3
F1		1
F2		
F3		1
F4		

(b)

Figure 5.2: Example mappings relevant to one task aspect.

of that particular information type but there is not any functionality that provides that information. By applying this method to every task aspect, we can determine what task aspects are not supported as well as why they are not supported. That is, the task aspects are unsupported because certain information types are unsupported by the functionalities of the computer aids.

The above two questions address what information types and, in turn, task aspects are not supported by the functionalities of the particular set of computer aids. Conversely, we can ask what functionalities of the computer aids are not actually used because they do not support any information types or task aspects. This is what the next two questions are about.

The third question, if a functionality of the computer aids does not support any information type, is the dual of the first question discussed above. There we obtain the answer by examining empty columns in the matrix of information types by functionalities of the computer aids. Similarly, we can check for empty rows in the same matrix in order to answer this inquiry. That the row of a functionality of the computer aids is empty indicates that this functionality does not support any information type.

The fourth question, if any functionality of the computer aids is not used by any task aspect, is equivalent to asking if all functionalities are used by some task aspects. This question is the dual of the second question. For each functionality we need to examine its connections with information types and, in turn, with task aspects. As an example, the subgraph relevant to functionality F4 in Figure 5.1(c) is shown in Figure 5.3(a) and its equivalent matrix in Figure 5.3(b). This matrix is obtained by selecting the F4 row of the matrix of information types by functionalities of the computer aids (where I5 is marked) and using this vector to “mask” the rows of the task aspects by information types matrix (where the I5 row



(a)

Information types	Task aspects			
	T1	T2	T3	T4
I5				

(b)

Figure 5.3: Example mappings relevant to one computer aids functionality.

is picked). An empty matrix indicates that all the information types provided by F4 are not used in any task aspect, that is, F4 does not support any task aspect.

5.3 The Configuration Problem

The precondition to the configuration problem is that every task aspect is supported by one or more functionalities of the computer aids. As demonstrated in the previous section, every functionality may not support all task aspects. For a certain task area, we would like to, ideally, be able to select a collection of functionalities that support all task aspects but do not include any superfluous functions. Such a collection is called *orthogonal* (DeMarco, 1979). Practically, we may not be able to find an orthogonal collection but we want to select a collection that contains the minimum of superfluous functions. Hereafter, we call a collection of functionalities of the computer aids selected for accommodating a certain purpose a *configuration*.

Consider that the profile of a functionality of the computer aids is a set containing those task aspects which the functionality can support. The configuration problem is to find the minimum number of functionalities such that the union of their profiles contains all task aspects. We decompose the configuration problem into two subproblems: the subconfiguration problem, for figuring out configurations that accommodate one task aspect, and the total configuration problem, for combining subconfigurations for all task aspects into one or more total configurations, which are the final solutions to the configuration problem.

5.3.1 The Problem of Deriving Subconfigurations

According to the two-mapping model in Section 5.1, functionalities of the computer aids support task aspects in terms of information types. The goal of the subconfiguration problem for a certain task aspect is to identify one or more configurations that accommodate that task aspect. By accommodating a task aspect we mean that the union of the information types supported by all functionalities in such a configuration contains the set of information types required for the task aspect and the number of functionalities included in the configuration is minimum. We call such a configuration a *subconfiguration* because it reflects only one task aspect, which is a part of the larger configuration problem.

Here is an operational description of the subconfiguration problem. To obtain a subconfiguration for a task aspect, we need to first obtain the subgraph of the two-mapping model that is relevant to the task aspect. We have described in the previous section the method of obtaining a matrix that is equivalent to the subgraph (see also Figure 5.2). Given this reduced matrix of information types by functionalities of the computer aids, to derive a subconfiguration is to find a minimum number of rows so that the overlay of those row vectors is a vector whose entries are all marked. The collection of functionalities represented by this minimum number of rows is the subconfiguration desired.

The problem of finding a subconfiguration is a *set covering* problem, which is NP-complete (Karp, 1972). This means that there is no efficient way for solving the problem and obtaining optimal solutions. Ad hoc methods, among other ways, are often used to generate some “reasonably good” solutions to an NP-complete problem. We propose in the next section an ad hoc method for coping with the subconfiguration problem. We term our method an *approximation method* in the

sense that it is not for generating optimal solutions but for generating some reasonably good solutions in an efficient manner.

In the following, we purposely relax the criterion of “minimal” number of functionalities of computer aids in a subconfiguration and take a looser criterion of “relatively low” number of functionalities in a subconfiguration. There are two reasons. The first is that the approximation method generates reasonably good solutions which may not be optimal. Relaxation of the criterion could increase the number of good solutions that can be obtained. The second is for the purpose of solving the total configuration problem. All minimal subconfigurations for all task aspects may not result in the best total configurations.

5.3.2 The Problem of Deriving Total Configurations

The total configuration problem is that of combining subconfigurations for all task aspects into one or more total configurations, which are the final solutions to the configuration problem. The term *total configuration* is used to distinguish from the term subconfiguration. Given the subconfigurations for all task aspects, the assignment of obtaining a total configuration is to select exactly one subconfiguration from the set of subconfigurations for each task aspect and form the union of the selected subconfigurations. The subconfigurations are selected in such a way that the union, which is the total configuration desired, contains a minimum number of functionalities of computer aids.

This following example illustrates the total configuration problem. Assume that we are dealing with task aspects T1, T2, and T3 and functionalities F1, F3, F4, and F6 of the computer aids, and that the subconfigurations for T1 are {F1, F3}, {F3, F4}, and {F1, F4, F6}, those for T2 are {F1, F4} and {F4, F6}, and those for T3 are {F1, F3, F4} and {F4, F6}. By selecting exactly one subconfiguration for

each task aspect and combining them together, we have 12 possible combinations: {F1, F3, F4}, {F1, F3, F4, F6}, {F1, F3, F4, F6}, {F1, F3, F4, F6}, {F1, F3, F4}, {F1, F3, F4, F6}, {F1, F3, F4, F6}, {F1, F3, F4, F6}, {F3, F4, F6}, {F1, F3, F4, F6}, {F1, F4, F6}, {F1, F3, F4, F6}, and {F1, F4, F6}. Three of these candidate configurations are better than others: {F1, F3, F4}, {F3, F4, F6}, and {F1, F4, F6}.

We call the problem of finding a total configuration the *minimum union* problem. It is an NP-complete problem. The NP-completeness of the minimum union problem can be verified by reducing the set covering problem to the minimum union problem. We propose in the next section an approximation method for coping with the total configuration problem. As in the case of the approximation method for the subconfiguration problem, the approximation method for the total configuration problem is intended for generating reasonably good solutions in an efficient manner.

5.4 Approximation Methods for Deriving Configurations

5.4.1 Approximation Method for Deriving Subconfigurations

Input to the approximation method for deriving subconfigurations is a matrix of information types by functionalities of computer aids (the columns are information types and the rows are functionalities) that is equivalent to the subgraph relevant to the task aspect under investigation. Outputs are a number of subconfigurations for the respective task aspect. Following is the procedure for deriving one subconfiguration, where S is the subconfiguration to be derived:

1. Set S to empty.
2. Exit if the matrix is empty.
3. Select the columns which have the least number of 1s.

4. Select the rows with the greatest number of 1s from all the rows corresponding to the 1s in the columns selected in Step 3.
5. If there is more than one eligible row, select the row(s) that will eliminate the least number of 1s (see Step 8).
6. If there is any tie, select one row arbitrarily.
7. Insert into S the computer aids functionality corresponding to the selected row.
8. Reduce the matrix by eliminating the selected row and all columns covered by the selected row.
9. Go to Step 2.

The procedure is demonstrated with the matrix given in Figure 5.4(a). First, select the columns with the least number of 1s; they are columns I1, I4, and I7. This means that we may select row F1, F3, F4, F5, or F6. Here we want to select the row that has the largest number of 1s. F4 and F5 are the candidates. The difference between choosing F4 and F5 is that F4 eliminates less 1s. Let us choose F4. After removing row F4 and associated columns, the matrix reduces to the one in Figure 5.4(b). Now column I1 has the least number of 1s and, therefore, we may choose between F1 and F6. Let us choose F1 because it has more 1s. After removing row F1 and associated columns, the matrix becomes empty and the procedure halts. Therefore, the subconfiguration {F1, F4} supports all the information types requisite for the particular task aspect with respect to the given matrix.

The procedure presented here derives one subconfiguration. If we trace down all choices at Steps 5 and 6, we will derive more than one subconfigurations. For example, if we choose F5 in addition to F4 in the above example, we will derive

Computer aids functionalities	Information types						
	I1	I2	I3	I4	I5	I6	I7
F1	1		1		1		
F2		1	1		1		
F3				1	1	1	
F4		1		1		1	1
F5		1	1			1	1
F6	1	1	1				

(a)

Computer aids functionalities	Information types		
	I1	I3	I5
F1	1	1	1
F2		1	1
F3			1
F5		1	
F6	1	1	

(b)

Figure 5.4: Demonstration of the subconfiguration method.

three more subconfigurations: {F1, F3, F5}, {F1, F4, F5}, and {F3, F5, F6}.

5.4.2 Approximation Method for Deriving Total Configurations

Inputs to the approximation method for deriving total configurations are sets of subconfigurations, one set for each task aspect. Outputs are a number of total configurations. Following is the procedure for deriving one total configuration, where C is the total configuration to be derived:

1. Set C to empty.
2. Exit if there is no subconfiguration to be considered.
3. If C contains any of the given subconfigurations, designate that subconfiguration as selected and go to Step 9.
4. Select the subconfigurations that contain the elements occurring most frequently in all subconfigurations but have the lowest cardinality.
5. If there is more than one eligible subconfiguration, select the one(s) whose buddy subconfigurations (for the same task aspect) have the highest cardinality.
6. If there is more than one eligible subconfiguration, select the one(s) whose intersection(s) with C have the highest cardinality.
7. If there is any tie, select one subconfiguration arbitrarily.
8. Merge the selected subconfiguration into C.
9. Remove the selected subconfiguration and all its buddy subconfigurations.
10. Go to Step 2.

The procedure is demonstrated with the subconfigurations given in Figure 5.5(a). F4 occurs most frequently. Those subconfigurations containing F4 and

T1	T2	T3
{F1, F3}	{F1, F4}	{F1, F3, F4}
{F3, F4}	{F4, F6}	{F4, F6}
{F1, F4, F6}		

(a)

T2	T3
{F1, F4}	{F1, F3, F4}
{F4, F6}	{F4, F6}

(b)

T2
{F1, F4}
{F4, F6}

(c)

Figure 5.5: Demonstration of the total configuration method.

having the lowest cardinalities are $\{F3, F4\}$, $\{F1, F4\}$, and $\{F4, F6\}$. Among them, $\{F3, F4\}$ and $\{F4, F6\}$ (for T3) have buddies of highest cardinalities. Let us choose $\{F3, F4\}$ (for T1). After removing T1, the remaining subconfigurations are those in Figure 5.5(b). Again, F4 occurs most frequently. Among those containing F4, $\{F1, F4\}$, $\{F4, F6\}$ (for T2), and $\{F4, F6\}$ (for T3) have the lowest cardinalities. Let us choose $\{F4, F6\}$ (for T3), because it has a buddy of the highest cardinality, and remove T3. Now the resultant configuration is $\{F3, F4, F6\}$. Since this configuration contains one of the subconfiguration for T2 (see Figure 5.5(c)), T2 is removed. There is no subconfiguration remaining and the procedure halts. The total configuration is thus $\{F3, F4, F6\}$.

As in the case of deriving subconfigurations, we may trace down all choices during the above procedure in order to derive multiple total configurations. If we follow the alternatives at Steps 5, 6, and 7 in the above example, we will also obtain these total configurations: $\{F1, F3, F4\}$ and $\{F1, F4, F6\}$.

5.4.3 Remarks on the Approximation Methods

Approximation methods are usually problem specific. This characteristic is reflected in the forms of the approximation methods presented above. Even knowing that the set covering problem can be reduced to the minimum union problem (that is, the former is subordinate to the latter), we still need separate approximation methods for each of them and these methods are tied to the problem representations. Namely, if we identify another problem to be a set covering problem in nature, we will need yet another approximation method for that new problem.

The rationale for using approximation methods to handle the subconfiguration problem and the total configuration problem is that of practicality. The

NP-completeness of the two problems demands that we find an efficient way for dealing with them. In some circumstances, as we mentioned in Section 5.3.1, we may want to relax the criteria for selection and do not need the results to be optimal. These circumstances also justify the use of approximation methods.

An approximation algorithm for the set covering problem was proposed by Johnson (1974). In terms of our matrix representation, the only strategy used in Johnson's algorithm is to select the row that contains the most 1s. Using this strategy alone can easily result in unacceptable answers. Our approximation method is superior in that we use two additional strategies—selecting the columns that contain the least number of 1s and selecting the rows that eliminate the least number of 1s—to improve the accuracy of the results.

5.5 Concluding Remarks

We have proposed a two-mapping model for describing the relations between task aspects, information types, and functionalities of computer aids, based on the idea that a classification of information types is the appropriate means of mediating a task area and a set of computer aids under study. We have presented methods, which are based on the two-mapping model, for answering the questions regarding usability of the computer aids to the task area. Another use of the two-mapping model is for examining the configuration problem of the computer aids with respect to a task area.

The problem of configuration arises because all functionalities of the computer aids may not support all aspects of a task area. We want to select a number of functionalities which together can support all task aspects but do not include many superfluous functions. We have decomposed the configuration problem into

a subconfiguration problem and a total configuration problem. The former addresses deriving configurations for a task aspect. The latter is concerned with deriving configurations for all task aspects. We have asserted that these problems are NP-complete in nature and have proposed effective approximation methods for deriving subconfigurations and total configurations, respectively.

In the next chapter, the two-mapping model for fitting computer aids and the methods for solving the usability and the configuration problems will be applied to investigate the use of electronic meeting systems, especially the GroupSystems electronic meeting system, in the task area of the group approach to information system requirements determination.

CHAPTER 6

FITTING ELECTRONIC MEETING SYSTEMS

The purpose of this chapter is to investigate the use of electronic meeting systems to assist in the group approach to information system requirements determination. The electronic meeting system being studied is the GroupSystems developed at The University of Arizona. We first instantiate the three dimensions of the model for fitting computer aids presented in the previous chapter: information types, task aspects, and computer aids functionalities, where the task aspects result from our analysis in Chapters 2, 3, and 4 and the functionalities of computer aids are the tools of GroupSystems. With the dimensions and the mappings between the dimensions instantiated, we then study the usability and the configuration problems of GroupSystems tools with respect to the task area by applying the methods proposed in the previous chapter. We want to learn whether or not GroupSystems tools can aid in the task area and how to combine GroupSystems tools to accommodate the task area.

6.1 Classification of Information Types

The key to realizing the approach for fitting computer aids presented in the previous chapter is the classification of information types. The need is a comprehensive classification that can be used to describe task aspects and to characterize computer aids functionalities. Essentially, the *granularity* of the information types of the classification should be large enough that the information types do carry

meanings and the *resolution* should be high enough that the information types are distinguishable from each other. Referring to the illustration in Figure 6.1, a desired classification of information types is one that is in quadrant I. With both large granularity and high resolution, such a classification is desired because we can do meaningful inferences with it.

Huber (1984) suggested that group decision support systems should provide support for numeric, textual, and relational types of information. This classification of information types belongs to quadrant II in Figure 6.1 because we consider the resolution to be high but the granularity to be too small. It is easy to distinguish between, for example, a piece of numeric information and a piece of textual information, but each information type does not carry much meaning in the sense that it is virtually impossible to measure the usefulness of the information type to aspects of a task under study.

Rouse and Cody (Rouse, 1986; Rouse & Cody, 1988) have proposed categorizing computer-aided design (CAD) and computer-aided engineering (CAE) systems in terms of the types of information supported by those computer aids. The information types are *facts*, *fantasies*, *feedback*, and *fellowship*. Facts are factual information mainly for purpose of reference. Information for fantasies can be provided by enabling *generation*, *integration*, and *elaboration* of solutions during the design process, where generation is about bringing in new ideas, integration is about fitting ideas together, and elaboration is about detailing ideas. Feedback information, including *monitoring*, *evaluation*, and *advice*, is the assistance and guidance that computer aids can provide the user. Monitoring is the “bookkeeping” information such as maintaining the relationships and checking the consistency among alternative designs. Evaluation is the information about the quality of alternatives. Advice is the ultimate feedback that guides the user through the design

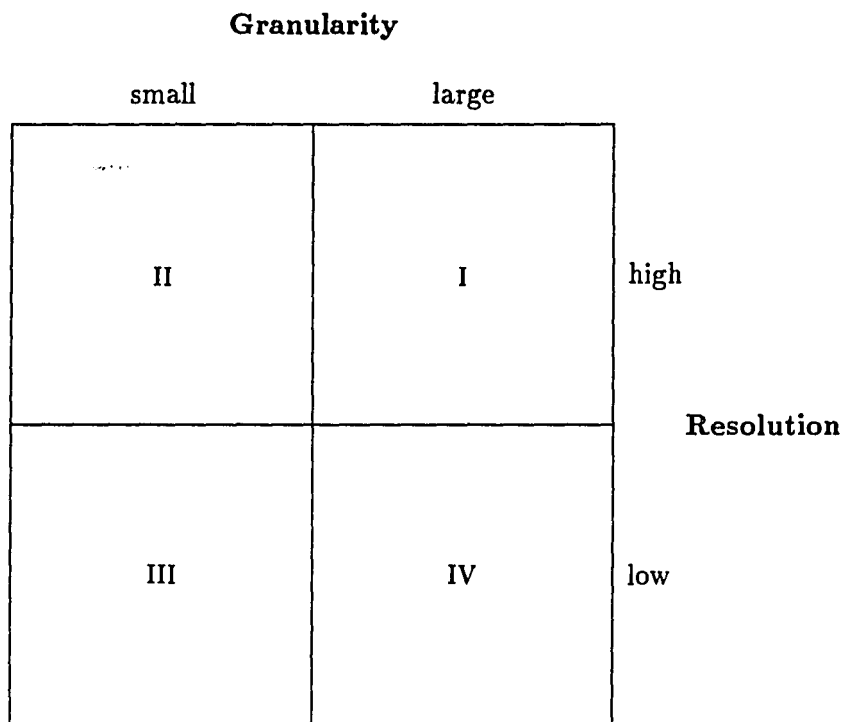


Figure 6.1: Granularity and resolution of information types classification.

process. Finally, fellowship concerns the group interaction happening during the design process.

Generally speaking, the Rouse–Cody classification of information types is comprehensive in that most other classifications of information or knowledge types¹ are somehow its subsets. Rasdorf's (1985) classification is an example of those subsets. The Rouse–Cody classification belongs to quadrant IV of the dichotomy shown in Figure 6.1. We consider the granularity to be large because the information types carry adequate meanings, while the resolution is low because the distinctions between the information types are subject to interpretation.

Based on the assumption that most kinds of computer aids (CAD, CAE, CASE, etc.) must have commonalities in the types of information supported, we adapt the Rouse–Cody classification for instantiating the information types classification dimension of the model for fitting computer aids. We have to enhance this classification and give each information type of this classification an adequate interpretation of our own when we use it to characterize task aspects and functionalities of computer aids. For one thing, the resolution of this classification is moderate; for another, we want to augment this classification so that it can accommodate the task area of information system requirements determination.

In particular, we decompose the monitoring and evaluation information types of the feedback category into more details. According to the types of relationships it can support, monitoring is decomposed into *item*, *list*, *tree*, *mesh*, and *plex*.² An item is an unstructured piece of information. A list is a group of items in

¹We take Tanimoto's (1987) interpretation that knowledge is information in context. Since we consider only classifications in the context of our research area, there is not much difference between the term "information classification" and the term "knowledge classification".

²This subclassification is adapted from Ho and Nunamaker's (1974) "data structure class" for

a sequential arrangement. A tree is a group of items in a hierarchical arrangement. A mesh is a group of items in a network, matrix, or equivalent arrangement. A plex is a group of items arranged in arbitrary manner. Evaluation is the kind of support that provides information about the quality of design alternatives. We decompose it into *dynamic evaluation* support and *static evaluation* support, depending on whether the evaluation is about the dynamic or the static properties of the design alternatives.

The enhanced classification of information types has large granularity, which is inherited from the Rouse–Cody classification, and high resolution, which is due to the decomposed information types. This suggests that a way to increase both granularity and resolution at the same time is to use a hierarchically layered classification. Viewing the classification at a higher layer, the granularity is large; viewing it at a lower layer, the resolution is high.

6.2 Information Support in Requirements Determination

The task aspects for instantiating the task aspects dimension of the model for fitting computer aids are those of the group approach to information system requirements determination. As we have concluded from Chapters 2, 3, and 4, those task aspects include:

- Design problem-solving behaviors
 - information collection
 - interpersonal communication
 - top-down decomposition

requirements statement language. The Ho–Nunamaker classification is more complicated and uses some different terms.

- bottom-up composition
 - making notes
 - labeling and retrieving models
 - simulation
- Structures and their constituents
 - problem structure
 - solution structure
 - Group process
 - identifying and resolving issues

Given instances of both the task aspects dimension and the information types dimension, we can instantiate the mapping between them. The instantiated mapping is shown in matrix format in Figure 6.2.

The information collection behavior (T1) is about collecting factual information, which is mainly for purposes of reference. This is what the facts information type (I1) can provide. Besides, fellowship type of information support (I13) helps collect information from other participants of the group. There could be many kinds of interpersonal communication behavior that could occur at any time and any place. Within the scope of our study, we concentrate on “computer-aided means” of interpersonal communication (T2), which is facilitated by the fellowship type of information support (I13).

Design problem-solving is a mixture of top-down decomposition behavior (T3) and bottom-up composition behavior (T4). In top-down decomposition, topics are expanded into more details and therefore the elaboration type of information

Information types	Task aspects									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
I1	1									
I2			1	1						
I3				1						1
I4			1							1
I5					1	1		1	1	1
I6					1	1		1	1	1
I7					1	1		1	1	1
I8					1	1		1	1	1
I9					1	1			1	1
I10							1			1
I11							1			1
I12										
I13	1	1								1

Information types		Task aspects	
Facts	I1	Design problem-solving behaviors	
Fantasies		information collection	T1
generation	I2	interpersonal communication	T2
integration	I3	top-down decomposition	T3
elaboration	I4	bottom-up composition	T4
Feedback		making notes	T5
monitoring		labeling and retrieving models	T6
item	I5	execution	T7
list	I6	Structures and their constituents	
tree	I7	problem structure	T8
mesh	I8	solution structure	T9
plex	I9	Group process	
evaluation		identifying and resolving issues	T10
dynamic evaluation	I10		
static evaluation	I11		
advice	I12		
Fellowship	I13		

Figure 6.2: The mapping between task aspects and information types.

support (I4) is needed. In bottom-up composition, topics are merged into consolidated ones and therefore the integration type of information support (I3) is needed. In both cases, the support of the generation information type (I2) helps bring in new ideas and keep the process going on.

In Section 3.3 we have suggested that multiple-channel hierarchy, which is a mesh structure, is an adequate organization for problem structure. Therefore, the problem structure (T8) needs the support of mesh (I8) and other simpler types of information. We have also suggested the use of the federation approach, which results in plex structures, in solution structure. The solution structure (T9) thus needs the support of plex (I9) and other simpler types of information.

The behavior of making notes in problem and solution structures (T5) is for controlling the progress of top-down decomposition or bottom-up composition. This behavior is assisted by the explicit use of problem and solution structures. By the same token, the behavior of labeling and retrieving models (T6) also benefits from the explicit use of problem and solution structures. Therefore, these two behaviors need the same degree of information support as problem structure (T8) and solution structure (T9).

During the progress of top-down decomposition and bottom-up composition, the simulation behavior (T7) helps examine the correctness of the results accumulated. This purpose can be achieved by dynamic evaluation (I10) as well as static evaluation (I11). The former type of support supplies evaluation information obtained by execution; the latter type of support supplies evaluation information obtained without execution.

The group process of identifying and resolving issues (T10) has two stages: identifying issues and resolving issues. As we have described in Section 4.2, the process of identifying issues involves examining the problem and solution struc-

tures by group participants. This needs not only the support of the fellowship information type (I13) but also that of the monitoring (including I5, I6, I7, I8, and I9) and evaluation (including I10 and I11) information types. The process of resolving issues usually involves adjusting the problem and solution structures. The support of the elaboration (I4) and integration (I3) information types is thus needed.

6.3 GroupSystems as Computer Aids

The motivation for studying GroupSystems in this research is two-fold. First, since GroupSystems intrinsically facilitates group meeting processes, it is a natural candidate to answer the need for computer aids for the group approach to information system requirements determination. Second, for a new technology such as GroupSystems, we usually want to know whether it is useful for the types of tasks to which we intend to apply it. The particular task which we study here is information system requirements determination. Results of this study may be used as a basis for improving GroupSystems and for studying the use of GroupSystems in other applications.

An installation of GroupSystems includes a meeting room, furniture, lighting, audio/video facilities, a number of workstations connected by a local area network, and a number of software tools³ running on the networked workstations (Dennis et al., 1988; Vogel et al., 1988). These tools perform in concordance with the other components of the installation to provide support for a group's meeting processes. It is possible to use many different tools in different sequences during a

³The software of the GroupSystems electronic meeting system is copyrighted by The University of Arizona, 1988, 1989, and by Ventana Corporation, 1990.

meeting session. Detailed descriptions of the tools and their usages can be found in the document by Ventana Corporation (1990).

In instantiating the dimension of functionalities of computer aids of the model for fitting computer aids, we choose to use “tool” as the unit of functionalities of the GroupSystems software because a tool is an intrinsic division of the GroupSystems software. That a tool can be used individually or in combination with other tools coincides with our understanding of the property of a computer aids functionality, as having been discussed in the analysis of the configuration problem in Section 5.3. We use the following tools as instances of the functionalities of computer aids of the fitting model: Briefcase (the File Reader mini-tool), Electronic Brainstorming, Idea Organization, Topic Commenter, Vote, Alternative Evaluation, Policy Formation, Group Outliner, Group Matrix, Group Dictionary, Stakeholder Identification, Questionnaire, and Group Writer.

The mapping between the instantiated dimension of information types and the instantiated dimension of functionalities of computer aids is obtained by characterizing the selected tools in terms of the information types they can support. When characterizing the tools, we consider only their ordinary modes of usage. The instantiated mapping is shown in matrix format in Figure 6.3. It is noted that, although “computer-aided fellowship” is an intrinsic nature of most GroupSystems tools, we consider that fellowship (information type I13) exists in those tools that let participants react directly to fellow participants’ particular inputs (for example, Electronic Brainstorming, F2) but not in those tools that do not permit direct reaction (for example, Vote, F5). Moreover, an ordered relation exists between the item, list, tree, mesh, and plex information types. When a tool supports a more complex information type, all less complex types are considered supported.

GroupSystems tools	Information types												
	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13
F1	1				1								
F2		1			1								1
F3		1	1		1	1							1
F4		1			1	1							1
F5											1		
F6					1	1					1		
F7			1	1	1								
F8		1	1	1	1	1	1						1
F9				1	1	1	1	1			1		
F10	1			1	1	1							
F11		1			1	1	1				1		
F12	1				1						1		
F13			1	1	1								1

GroupSystems tools	Information types
Briefcase	Facts I1
File Reader	F1 Fantasies
Electronic Brainstorming	F2 generation I2
Idea Organization	F3 integration I3
Topic Commenter	F4 elaboration I4
Vote	F5 Feedback
Alternative Evaluation	F6 monitoring
Policy Formation	F7 item I5
Group Outliner	F8 list I6
Group Matrix	F9 tree I7
Group Dictionary	F10 mesh I8
Stakeholder Identification	F11 plex I9
Questionnaire	F12 evaluation
Group Writer	F13 dynamic evaluation I10
	static evaluation I11
	advice I12
	Fellowship I13

Figure 6.3: The mapping between information types and GroupSystems tools.

From the viewpoint of tools, each row of the matrix in Figure 6.3 is the profile of a tool with respect to the classification of information types. Following are explanations of the matrix from the viewpoint of tools.

- File Reader (F1) supplies facts (I1) because factual information can be stored in files which are accessible to participants through this tool. The type of information supplied is item (I5) because the files accessible through File Reader are plain files.
- Electronic Brainstorming (F2), beginning with a given topic, allows participants to directly comment on the inputs of fellow participants. The results of this process are many new ideas, which we consider to be generation type information (I2). These ideas are stored in plain files of item information type (I5). Also, participants' direct reactions enable fellowship (I13).
- Idea Organization (F3) provides facilities for obtaining lists of items through group interaction. The effects are the generation (I2), item (I5), list (I6), and fellowship (I13) types of information support. This tool also fosters integration type information (I3) because it lets participants categorize data under item lists.
- Topic Commenter (F4), compared to Electronic Brainstorming (F2), is also a tool that provides generation (I2), item (I5), and fellowship (I13) types of information support because it lets participants comment on each other's inputs. The difference is that Topic Commenter begins with a given list of topics, which is of list information type (I6), for participants to categorize their inputs.
- Vote (F5) provides a number of voting methods for polling participants'

opinions on one or more items. Voting results are static evaluation type of information (I6). Vote does not support item or list information type because it does not *maintain* any of them.

- Alternative Evaluation (F6) allows participants to evaluate a list of items against another list of items. The resultant information is of evaluation type (I6). This tool, in contrast to Vote (F5), does maintain the list (I6) and item (I5) information types.
- Policy Formation (F7) is used iteratively to allow participants to send in refined versions of a statement about a topic. A newer version of the statement may be a more integrated one or a more elaborated one. The information types supported are therefore integration (I3) and elaboration (I4). The statement produced is of the item information type (I5).
- Group Outliner (F8) provides a hierarchical structure for organizing topics. Maintenance of the hierarchical structure is the support of the tree (I7) and, consequently, the list (I6) and item (I5) types of information. That participants can react directly to each other's inputs, as in the case of Electronic Brainstorming (F2), indicates that this tool supports the generation (I2) and fellowship (I13) types of information. In addition, manipulation of the hierarchical structure helps split or merge topics. This is considered the support of elaboration (I4) and integration (I3) types of information.
- Group Matrix (F9) provides a matrix structure of which each dimension is a list of items. Maintenance of the matrix structure is the support of the mesh (I8), tree (I7), list (I6), and item (I5) types of information. Participants establish the relationship between the two lists of items by filling in appro-

priate information in the cells of the matrix. This is considered the support of elaboration type of information (I4). The same structure and process can also be used to compare items of the two lists. This is static evaluation (I11) type of support.

- Group Dictionary (F10) helps a group of participants create a common terminology. The list of words and their definitions derived by using this tool become factual information that can be referenced later on. This tool thus supports facts (I1), list (I6), and item (I5) types of information. The process of deriving the definition of an item is also considered a support of elaboration type of information (I4).
- Stakeholder Identification (F11) is used to create a list of items and, for each item in the first list, create a secondary list of items. This is a process of generation that results in a hierarchy of items. Therefore, the generation (I2) as well as the tree (I7), list (I6), and item (I5) information types are supported. This tool is then used to analyze the impact of each secondary list of items to its superior item. This is a static evaluation type of information support (I11).
- Questionnaire (F12) is used to solicit information from the participants by sending them questionnaires to fill out. When the information collected is factual information, the support is of facts information type (I1); when the information collected is measurement, the support is of static evaluation information type (I11). In either case, the information is of item type (I5) because it is stored in plain files.
- Group Writer (F13) provides a shared text editor with which participants can

directly and simultaneously manipulate a text file. Results of the manipulation may be more elaborated or more integrated. Thus, this tool provides fellowship (I13), elaboration (I4), and integration (I3) types of information support. The text files handled by this tool are of item information type (I5).

6.4 The Use of GroupSystems in Requirements Determination

Given the instantiated mappings between task aspects, information types, and GroupSystems tools presented in Figures 6.2 and 6.3, we are in a position to study the usability and configuration problems in the context of the use of GroupSystems tools in the task area of the group approach to information system requirements determination. We proceed by applying to the instantiated mappings the methods proposed in the previous chapter.

6.4.1 The Usability Problem

The usability problem of computer aids with respect to a task area includes four questions (see Section 5.2). The first question, if any information type is not supported by any GroupSystems tool, can be answered by examining the columns of the matrix of information types by GroupSystems tools in Figure 6.3. There are three empty columns in that matrix: column I9, for the plex information type; column I10, for the dynamic evaluation information type; and column I12, for the advice information type. This means that these three information types are not supported by any GroupSystems tools.

To answer the second question, if any task aspect is not supported by GroupSystems tools, we have to check for each task aspect if it is supported by

some GroupSystems tools. For task aspect T5, we extract column T5 (where I5, I6, I7, I8, and I9 are marked) from the matrix of task aspects by information types in Figure 6.2 and use this vector to “mask” the columns of the matrix of information types by GroupSystems tools in Figure 6.3. The resultant matrix contains five columns, of which column I9 is empty. This means that task aspect T5 is not supported by GroupSystems tools because information type I9 is not supported. By following the same procedure for all other task aspects, we conclude that: task aspects T5, T6, and T9 are not supported because information type I9 (plex) is not supported; task aspect T7 is not supported because information type I10 (dynamic evaluation) is not supported; and task aspect T10 is not supported because the support of both I9 and I10 are wanting.

The third question, if there is any GroupSystems tool that does not support any information type, can be answered by examining the rows of the matrix of information types by GroupSystems tools in Figure 6.3. Every row has at least one marked entry. Therefore, every GroupSystems tool does support some information types.

The fourth question is if any GroupSystems tool is not used by any task aspect, or if every GroupSystems tool is possibly used by some task aspects. We have to check for each tool that it does support some information types and, in turn, these information types are used by some task aspects. For tool F1, we extract row F1 (where I1 and I5 are marked) from the matrix of information types by GroupSystems tools in Figure 6.3. That this vector is not empty indicates that F1 does support some information types. We then use this vector to “mask” the rows of the matrix of task aspects by information types in Figure 6.2 and obtain a matrix which is not empty. This non-empty matrix indicates that GroupSystems tool F1 does support some task aspects. We repeat this procedure for each tool

and conclude that every GroupSystems tool is possibly used to support some task aspects because every tool supports some information types that are useful to some task aspects.

In all, the answers to the usability questions show that every GroupSystems tool is possibly used to support some task aspects but five task aspects are not supported by GroupSystems tools. This is because the I9 (plex) and I10 (dynamic evaluation) information types are not supported by any GroupSystems tools. In order to use GroupSystems to assist in our task area, we need to supplement the GroupSystems tool set with some tools that can provide these two types of information. A candidate tool for providing the plex information type (and other less complex types) is called Enterprise Analyzer, which runs on the same network platform as GroupSystems tools do (*PLEXSYS user guide*, 1989). This tool helps maintain a global plex structure that can be expanded and shrunk by users. For the information support of dynamic evaluation, we envision that an Animation tool is needed. To be more useful, this tool should also provide the support of static evaluation.

The advice information type (I12) is also not supported by any GroupSystems tools. This is understandable because no GroupSystems tool provides any guidance to users. In other words, people who use GroupSystems are active and GroupSystems tools are passive. By crosschecking the mapping between task aspects and information types (Figure 6.2), we find that the advice information type is not used by any task aspect. That is, the advice information type is, in fact, not required. The reason is that the advice information type signifies a drastically different form of using computer aids where computer aids play an active role; but we did not consider that kind of influence when we analyzed the task area in the previous chapters. This understanding indicates that the advice information

type, though not essential for current computer aids, should become vital to future computer aids that provide for better functionalities.

6.4.2 The Configuration Problem

The configuration problem is to select a minimum number of tools from the GroupSystems tool set to form configurations that accommodate the task area of the group approach to information system requirements determination. The precondition is that every task aspect is possibly supported by some GroupSystems tools. We assume that the Enterprise Analyzer and the Animation tools mentioned above are in the GroupSystems tool set. The extended mapping between information types and GroupSystems tools is in the matrix in Figure 6.4. The approximation methods for deriving subconfigurations and total configurations (see Section 5.4) are applied to the mappings in Figures 6.2 and 6.4.

We first have to obtain subconfigurations for every task aspect. For task aspect T10, we extract column T10 (where I3, I4, I5, I6, I7, I8, I9, I10, I11, and I13 are marked) from the matrix of task aspects by information types in Figure 6.2 and use this vector to “mask” the columns of the matrix of information types by GroupSystems tools in Figure 6.4. This results in a ten-column (I3, I4, I5, I6, I7, I8, I9, I10, I11, and I13) matrix for input to the approximation method for deriving subconfigurations. Following the approximation method, we choose columns I9 and I10 because they contain the least number of 1s. From the rows corresponding to the 1s in I9 and I10 (including F14 and F15), we select F14 because it covers the greatest number of 1s. After eliminating row F14 and its related columns, five columns (I3, I4, I10, I11, and I13) remain. In the second iteration, column I10 contains the least number of 1s and the only row corresponds to I10 is F15. We select F15. After eliminating row F15 and related columns, three columns (I3, I4,

GroupSystems tools	Information types												
	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13
F1	1				1								
F2		1			1								1
F3		1	1		1	1							1
F4		1			1	1							1
F5											1		
F6					1	1					1		
F7			1	1	1								
F8		1	1	1	1	1	1						1
F9				1	1	1	1	1			1		
F10	1			1	1	1							
F11		1			1	1	1				1		
F12	1				1						1		
F13			1	1	1								1
F14					1	1	1	1	1				
F15										1	1		

GroupSystems tools		Information types	
Briefcase		Facts	I1
File Reader	F1	Fantasies	
Electronic Brainstorming	F2	generation	I2
Idea Organization	F3	integration	I3
Topic Commenter	F4	elaboration	I4
Vote	F5	Feedback	
Alternative Evaluation	F6	monitoring	
Policy Formation	F7	item	I5
Group Outliner	F8	list	I6
Group Matrix	F9	tree	I7
Group Dictionary	F10	mesh	I8
Stakeholder Identification	F11	plex	I9
Questionnaire	F12	evaluation	
Group Writer	F13	dynamic evaluation	I10
Enterprise Analyzer	F14	static evaluation	I11
Animation	F15	advice	I12
		Fellowship	I13

Figure 6.4: The extended mapping between information types and GroupSystems tools.

and I13) remain. In the third iteration, column I3 contains the least number of 1s. Rows F8 and F13, which correspond the 1s in column I3, contain the greatest number of 1s. We may choose F8 or F13. In either case, the matrix becomes empty after this iteration. Therefore, the subconfigurations for task aspect T10 are {F8, F14, F15} and {F13, F14, F15}. The same procedure is applied to other task aspects. The resultant subconfigurations are shown in Figure 6.5(a).

The subconfigurations are fed into the approximation method for total configurations. In all subconfigurations, F8 occurs most frequently. Among all subconfigurations that contain F8, {F8} has the lowest cardinality. We put {F8} into the total configuration and remove all task aspects that have {F8} as their subconfigurations. For the remaining task aspects (T1, T5, T6, T7, T8, T9, and T10), F14 occurs most frequently in their subconfigurations. Among all subconfigurations that contain F14, {F14} has the lowest cardinality. We merge {F14} into the total configuration (now it is {F8, F14}) and remove all task aspects that have {F14} as their subconfigurations. Task aspects T1, T7, and T10 remain. F1, F10, and F12 occur most frequently in all remaining subconfigurations. All the eligible subconfigurations containing F1, F10, and F12 have the same cardinalities and all their buddy subconfigurations have the same cardinalities. Therefore, among the eligible subconfigurations we need to select the ones which have the most overlap with the total configuration. {F1, F8}, {F8, F10}, and {F8, F12} (all for T1) are candidates. Let us choose {F1, F8} and merge it into the total configuration. Now the total configuration is {F8, F14, F1}. After removing T1, T7 and T10 remain and F15 occurs most frequently in all their subconfigurations. Among the subconfigurations containing F15, {F15} has the lowest cardinality. We merge {F15} into the total configuration, which becomes {F8, F14, F1, F15}, and remove T7. Now that the total configuration contains one of the subconfigurations for the re-

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
{F1, F2}	{F2}	{F8}	{F3}	{F14}	{F14}	{F15}	{F9}	{F14}	{F8, F14, F15}
{F1, F3}	{F3}		{F8}				{F14}		{F13, F14, F15}
{F1, F4}	{F4}								
{F1, F8}	{F8}								
{F1, F13}	{F13}								
{F2, F10}									
{F2, F12}									
{F3, F10}									
{F3, F12}									
{F4, F10}									
{F4, F12}									
{F8, F10}									
{F8, F12}									
{F10, F13}									
{F12, F13}									

(a)

{F8, F14, F15, F1}
 {F8, F14, F15, F10}
 {F8, F14, F15, F12}

(b)

Task aspects		GroupSystems tools	
Design problem-solving behaviors		Briefcase	
information collection	T1	File Reader	F1
interpersonal communication	T2	Electronic Brainstorming	F2
top-down decomposition	T3	Idea Organization	F3
bottom-up composition	T4	Topic Commenter	F4
making notes	T5	Vote	F5
labeling and retrieving models	T6	Alternative Evaluation	F6
execution	T7	Policy Formation	F7
Structures and their constituents		Group Outliner	F8
problem structure	T8	Group Matrix	F9
solution structure	T9	Group Dictionary	F10
Group process		Stakeholder Identification	F11
identifying and resolving issues	T10	Questionnaire	F12
		Group Writer	F13
		Enterprise Analyzer	F14
		Animation	F15

Figure 6.5: Configurations of GroupSystems tools.

maining task aspect T10, we remove T10. No more subconfigurations remain and the procedure halts.

In the above procedure, if we choose {F8, F10} or {F8, F12} for T1, we can derive an additional total configuration in each case. In total, we have three total configurations, which are shown in Figure 6.5(b). This result indicates that we can use any of these three configurations of GroupSystems tools to aid in the group approach to information system requirements determination.

The three admissible total configurations listed in Figure 6.5(b) are very similar. They all contain tools F8, F14 and F15, and they differ in only one tool. Referring to the extended matrix of information types by GroupSystems tools in Figure 6.4, F14 and F15 are the only tools that provide information types I9 and I10, respectively. It is certain that these two tools are included in the total configurations. Not considering I12 and those information types supplied by F14 and F15, F8 provides more information types than other tools do. The only information type that is not provided by F8 is then fulfilled by tool F1, F10, or F12. This explanation demonstrates that our approximation methods are capable of deriving adequate results.

That the subconfigurations for all task aspects are obtained in no more than three iterations of the subconfiguration method and the total configurations are obtained in five iterations of the total configuration method confirms that the two approximation methods can quickly arrive at solutions.

6.5 Concluding Remarks

In order to examine the use of the GroupSystems electronic meeting system in the task area of the group approach to information system requirements, we

have instantiated the three dimensions of the model for fitting computer aids. We first adapted an information types classification that has adequate granularity and resolution and then built up the mappings between information types and the other two dimensions, namely, task aspects and GroupSystems tools.

By applying to the instantiated model the methods for answering the usability questions, we have gained insights about the usability of GroupSystems tools in the task area. These insights include the facts that all tools are possibly used to support some task aspects and that some task aspects are not supported by the tools because certain requisite information types are not supported. Consequently, we have suggested what tools might have to be supplied in order to provide adequate computer aids for the task area. By applying the approximation methods for deriving configurations, we have also obtained a number of configurations of GroupSystems tools for use in the task area. The process of applying the approximation methods revealed that they can effectively and efficiently conclude to solutions.

CHAPTER 7

USING GROUPSYSTEMS IN REQUIREMENTS DETERMINATION

In order to demonstrate the usefulness of our model for fitting computer aids and our approximation methods for deriving tool configurations, we report in this chapter the account of an authentic case of using GroupSystems in requirements determination. We first present the background of the case and then analyze the use of GroupSystems tools in terms of the information types involved in each meeting session of the case.

7.1 Background of the Case

The U.S. Army currently spends five billion dollars annually and employs over 250,000 people to operate the many Army, National Guard, and Reserve installations around the world. The Army Installation Support Module (ISM) Project has been initiated in an attempt to improve the productivity of managing installations. The objective of the \$172-million ISM Project has been to develop an information system that will be globally installed at all installations to support the management of those installations. This information system, consisting of a number of modules, will assist in the management of the various activities that occur at each installation by supporting all standard activities required by regulations as well as any local requirements instituted by the installation commander. This Army-wide information system should solve the current problem of no common method for managing installations and the related problems of information redun-

dancies, poor communication and coordination, and non-standardization between installations and between functional areas at a single installation.

The task of deriving requirements for the ISM Project required the involvement of personnel from many functional areas and from the many installations in all regions of the world. In order to handle the complexity of involving many personnel, the Army decided to use GroupSystems to assist in this task. Representatives from the Army have been selected to use GroupSystems to define requirements for modules of the ISM Project. The requirements of one or two modules were obtained during each meeting session. For each module, the objective of using GroupSystems has been to gain an understanding of the activities involved in that module and to derive a functional description of the system to be developed for that module.

Six meeting sessions were conducted from February, 1990 to February, 1991. The requirements of eight modules were obtained during the six meeting sessions. The duration of each meeting session ranged from one week to three weeks. The participants of the meeting sessions varied from session to session. Those participated in a meeting session were representatives from all the major geographic commands and the functional commands and from the functional areas relevant to the module being developed. These meeting sessions were generally regarded by the participants as successes because, if traditional techniques were used, the same amount of results would have taken much longer time to derive and the quality of the results would not be comparable. Details of the meeting sessions have been reported in Daniels, Dennis, Hayes, Nunamaker, and Valacich (1991) and Hayes et al. (1990).

In the following we concentrate our analysis on the five meeting sessions that were held at The University of Arizona, while omit the one that was held

elsewhere. For reason of convenience, we will identify the meeting sessions by numbers instead of by the dates on which they took place.

7.2 Analysis of the Meeting Sessions

Shown in Figure 7.1 is a chronicle of the tools used in the five meeting sessions. Some additional tools, including some tools tailored for the ISM meeting sessions and some commercially available products, were occasionally used in conjunction with GroupSystems tools. In meeting session 1, WordPerfect was used by individuals to edit portions of the functional description and Lotus 1-2-3 was used to maintain a list of items. dBase III+ was used in meeting session 2 to maintain lists of items. The Group Case used in meeting session 4 was a canned database program for maintaining lists of items. In meeting session 5, PC-Write was used to create a list of items. FD Editor was a revised version of Group Writer and EA Process Editor and EA Form Editor were two canned database programs for maintaining lists of items.

By following arguments similar to those presented in Section 6.3, we determine that the information types supported by these additional tools are as following:

- Group Case, EA Process Editor, EA Form Editor, Lotus 1-2-3, and dBase III+ support item and list (I5 and I6) information types.
- FD Editor supports the same information types as Group Writer, namely, integration, elaboration, item, and fellowship (I3, I4, I5, and I13).
- WordPerfect supports integration, elaboration, and item (I3, I4, and I5) information types.

Session	Tools used
1	F2, F4, WP, 123
2	F2, dB, F9
3	F2, F2, F3, F5, F5, F2, F3, F13, F9, F2, F3, F5, F2, F3, F5, F2, F3, F5, F5, F3, F5, F5
4	F2, F1, F3, F5, GC, F4, F9, F4, F4, F13
5	F2, F3, F5, F5, PCW, FDE, PE, FE, F9, F3, FDE

GroupSystems tools

Briefcase

File Reader	F1
Electronic Brainstorming	F2
Idea Organization	F3
Topic Commenter	F4
Vote	F5
Alternative Evaluation	F6
Policy Formation	F7
Group Outliner	F8
Group Matrix	F9
Group Dictionary	F10
Stakeholder Identification	F11
Questionnaire	F12
Group Writer	F13

Additional GroupSystems tools

Group Case	GC
FD Editor	FDE
EA Process Editor	PE
EA Form Editor	FE

Other tools

WordPerfect	WP
Lotus 1-2-3	123
dBase III+	dB
PC-Write	PCW

Figure 7.1: Tools used in the meeting sessions of the case.

- PC-Write supports generation, integration, item, and list (I2, I3, I5, and I6) information types.

In the previous chapter, we derived three total configurations of GroupSystems tools that could be used in assisting in the requirements determination of information systems (see Figure 6.5). By referring to the matrix of information types by GroupSystems tools in Figure 6.4, we can tally the information types involved in those total configurations. For example, for the configuration {F8, F14, F15, F1}, we can obtain the set of information types involved in it by extracting the F8, F14, F15, and F1 rows of the matrix in Figure 6.4 and deriving the union of these rows. In fact, all the three total configurations cover the same set of information types. Let this set of information types be N . Then, $N = \{I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I13\}$.

In the same manner, we can derive the sets of information types involved in the five meeting sessions of the case. These sets, named $S1$ through $S5$, are listed along with N in Figure 7.2. Also listed in the same figure is S , which is the union of $S1$ through $S5$. S indicates all the information types involved in all five meeting sessions.

By comparing the sets listed in Figure 7.2, we see that all $S1$ through $S5$ are subsets of N and that the cardinalities of the differences between N and $S1$ through $S5$ are 6, 4, 3, 2, and 3. The first implication is that, when the administrator of the five meeting sessions was becoming more experienced, the sets of information types involved in the meeting sessions converged. The second implication is that our fitting model and configuration methods result in total configurations whose information type profiles are very close to the profiles of the tools used in authentic meeting sessions, although the administrator of these meeting sessions was not aware of the concepts and models developed in the research for this dissertation.

$N = \{I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I13\}$

$S1 = \{I2, I3, I4, I5, I6, I13\}$

$S2 = \{I2, I4, I5, I6, I7, I8, I11, I13\}$

$S3 = \{I2, I3, I4, I5, I6, I7, I8, I11, I13\}$

$S4 = \{I1, I2, I3, I4, I5, I6, I7, I8, I11, I13\}$

$S5 = \{I2, I3, I4, I5, I6, I7, I8, I11, I13\}$

$S = \{I1, I2, I3, I4, I5, I6, I7, I8, I11, I13\}$

Information types	
Facts	I1
Fantasies	
generation	I2
integration	I3
elaboration	I4
Feedback	
monitoring	
item	I5
list	I6
tree	I7
mesh	I8
plex	I9
evaluation	
dynamic evaluation	I10
static evaluation	I11
advice	I12
Fellowship	I13

Figure 7.2: Information types involved in the meeting sessions of the case.

The third implication is that our fitting model and configuration methods take into consideration more details than the rules-of-thumb used by the administrator of the authentic meeting sessions. This last argument explains why N covers all $S1$ through $S5$. We therefore are confident in our fitting model and approximation methods for deriving tool configurations.

7.3 Concluding Remarks

We have presented in this chapter an authentic case of using GroupSystems tools in requirements determination. By analyzing the information types involved in each meeting session of the case, we concluded that our model for fitting computer aids and our approximation methods for deriving tool configurations provide results that are close, yet superordinate, to the set of tools used in the case. This case thus demonstrates the usefulness of our fitting model and configuration methods. We will discuss in the next chapter ways to further improve the model for fitting computer aids and the approximation methods for solving the configuration problem.

CHAPTER 8

CONCLUSIONS

The research problem of this dissertation concerns the use of computer aids to facilitate the group process of information system requirements determination. Our approach, as we argued in Chapter 1, is to investigate the feasibility of applying existing computer aids to the problem, rather than developing an entirely new set of computer aids specifically tailored to requirements determination. The particular set of computer aids studied in this research is the GroupSystems electronic meeting system. This chapter summarizes the main points of this research and discusses the directions of future research.

8.1 Summary

In this research, we have achieved these goals:

- We developed a model for fitting a set of computer aids to a given task area in Chapter 5. This model describes the relations between task aspects, information types, and functionalities of the computer aids, based on the idea that a classification of information types is the appropriate means of mediating a task area and a set of computer aids. The major advantage of using two mappings (one between task aspects and information types and one between information types and functionalities) in this model is flexibility. One may use this model to study different computer aids or task areas by specifying different instances of functionalities and task aspects, respectively,

in this model.

- We analyzed in Chapter 5 the two parts of the problem of assessing the feasibility of applying a set of computer aids to a given task area: the usability problem and the configuration problem. With the matrix representation of the model for fitting computer aids, we developed methods, which involve examining appropriate rows and columns of the matrixes, for deriving answers to the usability problem. We also asserted that the configuration problem, including the subconfiguration problem and the total configuration problem, is NP-complete in nature. This means that there is no efficient way for solving the problem and obtaining optimal solutions.
- We also devised in Chapter 5 approximation methods for coping with the intractability of the subconfiguration and the total configuration problems. By manipulating the matrix representation of the model for fitting computer aids, the approximation methods can obtain reasonably good solutions in reasonable time.
- We analyzed in Chapters 2, 3, and 4 the task area of the group approach to information system requirements determination. We proposed a generational framework for modeling the information system development process and the object to be developed. We inferred that the process of requirements determination is a design problem-solving process involving a number of problem-solving behaviors applied to a problem structure and a solution structure that constitute the object. We judged that the kind of participation involving both technical and non-technical people in consensus-style process is suitable to requirements determination. And we outlined how a group of people can work together to derive a consensus model that contains

the problem and solution structures of the requirements of an information system. From these analyses we derived a list of aspects that characterize the task of the group approach to information system requirements determination.

- We instantiated in Chapter 6 the three dimensions of the model for fitting computer aids. We adapted a hierarchically layered classification of information types that has adequate granularity and resolution and then built up the mappings between information types and the other two dimensions, namely, aspects of the task of the group approach to requirements determination and tools of the GroupSystems electronic meeting system.
- We assessed the feasibility of using the GroupSystems electronic meeting system to aid in the group process of information system requirements determination in Chapter 6 by applying the methods for solving the usability and the configuration problems to the instantiated fitting model. We concluded that all GroupSystems tools are possibly used in some task aspects but some task aspects are not supported by the GroupSystems tool set because certain required information types are not supported. After suggesting appropriate new tools to supplement the GroupSystems tool set, we derived some configurations of GroupSystems tools for use in the group approach to information system requirements determination.
- We analyzed in Chapter 7 the information types involved in the meeting sessions of an authentic case of using GroupSystems tools in requirements determination. We demonstrated the usefulness of our model for fitting computer aids and approximation methods for deriving configurations by expounding the similarity between the information types involved in the meeting ses-

sions of the case and those involved in the configurations generated by the approximation methods.

The contribution of this research, on one side, is in the investigation into the general problem of fitting existing computer aids to task areas and, on the other side, is in the investigation into the specific problem of assessing the feasibility of using GroupSystems electronic meeting system in information system requirements determination.

8.2 Future Research

The question left unanswered in this research is, “which is the best configuration?” given a number of admissible total configurations generated by our approximation methods. This problem can be dealt with in three ways.

The first is to find better classifications of task aspects, information types, and functionalities of computer aids. Finer and more detailed classifications can help increase the accuracy of the outputs of the approximation methods.

The second is to perform evaluations of the approximation methods. Analytically, we may attempt to determine the bounds of their complexities. Empirically, we may test them with different patterns of data in order to gain insights into their behaviors.

The third is to supplement the approximation methods with additional capabilities to help in selection among a number of admissible configurations. This may involve the use of additional situational variables and the use of some inference mechanisms to sieve out the best configuration contingent on those variables.

REFERENCES

- Adelson, B., & Soloway, E. (1985). The role of domain experience in software design. *IEEE Transactions on Software Engineering, SE-11*, 1351–1360.
- Adelson, B., & Soloway, E. (1988). A model of software design. In M. T. H. Chi, R. Glaser, & M. J. Farr (Eds.), *The nature of expertise* (pp. 185–208). Hillsdale, NJ: Erlbaum.
- Akin, O. (1978). How do architects design? In J.-C. Latombe (Ed.), *Artificial intelligence and pattern recognition in computer aided design* (pp. 65–103). New York: North-Holland.
- Alexander, C. (1964). *Notes on the synthesis of form*. Cambridge, MA: Harvard University Press.
- Balzer, R. (1985). A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering, SE-11*, 1257–1268.
- Blanchard, B. S., & Fabrycky, W. J. (1990). *Systems engineering and analysis* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61–72.
- Byte staff. (1989). Making a case for CASE. *Byte, 14*(13), 154–158, 160–162, 164–166, 168, 170–171.
- Carroll, J. M., Thomas, J. C., & Malhotra, A. (1979). Clinical-experimental analysis of design problem solving. *Design Studies, 1*, 84–92.
- Carroll, J. M., Thomas, J. C., Miller, L. A., & Friedman, H. P. (1980). Aspects of solution structure in design problem solving. *American Journal of Psychology, 93*, 269–284.
- Cohen, B. (1982). Justification of formal methods for system specification. *Software and Microsystems, 1*, 119–127.
- Connor, D. (1985). *Information system specification and design road map*. Englewood Cliffs, NJ: Prentice-Hall.

- Daft, R. L., & Lengel, R. H. (1986). Organizational information requirements, media richness and structural design. *Management Science*, *32*, 554–571.
- Daft, R. L., Lengel, R. H., & Trevino, L. K. (1987). Message equivocality, media selection, and manager performance: Implications for information systems. *MIS Quarterly*, *11*, 355–366.
- Daniels, R. M., Jr., Dennis, A. R., Hayes, G., Nunamaker, J. F., Jr., & Valacich, J. (1991). Enterprise Analyzer: Electronic support for group requirements elicitation. In *Proceedings of the Twenty-fourth Annual Hawaii International Conference on System Sciences: Vol. 3* (pp. 43–52).
- Davis, A. M. (1988a). A comparison of techniques for the specification of external behavior of systems. *Communications of the ACM*, *31*, 1098–1115.
- Davis, A. M. (1988b). A taxonomy of the early stages of the software development life cycle. *The Journal of Systems and Software*, *8*, 297–311.
- Davis, G. B. (1982). Strategies for information requirements determination. *IBM Systems Journal*, *21*, 4–30.
- Davis, G. B., & Olson, M. H. (1985). *Management information systems: Conceptual foundations, structure, and development* (2nd ed.). New York: McGraw-Hill.
- DeMarco, T. (1979). *Structured analysis and system specification*. Englewood Cliffs, NJ: Prentice-Hall.
- Dennis, A. R., George, J. F., Jessup, L. M., Nunamaker, J. F., Jr., & Vogel, D. R. (1988). Information technology to support electronic meetings. *MIS Quarterly*, *12*, 591–624.
- Distaso, J. R. (1980). Software management—A survey of the practice in 1980. *Proceedings of the IEEE*, *68*, 1103–1119.
- Dubois, E., Hagelstein, J., & Rifaut, A. (1988). Formal requirements engineering with ERAE. *Philips Journal of Research*, *43*, 393–414.
- Friedman, A. L., & Cornford, D. S. (1989). *Computer systems development: History, organization and implementation*. Chichester, England: Wiley.
- Ginzberg, M. J. (1979). Participative system design. In N. Szyperski & E. Grochla (Eds.), *Design and implementation of computer-based information systems* (pp. 215–220). Alphen aan den Rijn, The Netherlands: Sijthoff & Noordhoff.

- Gorb, P., & Dumas, A. (1987). Silent design. *Design Studies*, 8, 150–156.
- Guindon, R., & Curtis, B. (1988). Control of cognitive processes during software design: What tools are needed? In E. Soloway, D. Frye, & S. B. Sheppard (Eds.), *Proceedings of CHI '88 Conference on Human Factors in Computing Systems* (pp. 263–268). New York: ACM.
- Guindon, R., Krasner, H., & Curtis, B. (1987). Breakdowns and processes during the early activities of software design by professionals. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), *Empirical studies of programmers: Second workshop* (pp. 65–82). Norwood, NJ: Ablex.
- Harandi, M. T. (1988). Building a knowledge-based software development environment. *IEEE Journal on Selected Areas in Communications*, 6, 862–868.
- Hayes, G., Dennis, A. R., Daniels, R. M., Jr., Ramesh, V., Nunamaker, J. F., Jr., Vogel, D., & Valacich, J. (1990). *Enterprise Analyzer: Electronic support for the system design team*. Tucson: University of Arizona, Karl Eller Graduate School of Management, Department of Management Information Systems.
- Hekmatpour, S., & Ince, D. (1988). *Software prototyping, formal methods and VDM*. Wokingham, England: Addison-Wesley.
- Ho, T. I. M., & Nunamaker, J. F., Jr. (1974). Requirements statement language principles for automatic programming. In *Proceedings of ACM Annual Conference* (pp. 279–288).
- Huber, G. (1984). Issues in the design of group decision support systems. *MIS Quarterly*, 8, 195–204.
- Hunt, R. M. (1987). The difficulties of design problem formulation. In W. B. Rouse & K. R. Boff (Eds.), *System design: Behavioral perspectives on designers, tools, and organizations* (pp. 145–157). New York: North-Holland.
- IBM Information Services. (1987). *Joint Application Design* (No. G520–0008–01). Milford, CT: Author.
- Ives, B., & Olson, M. H. (1984). User involvement and MIS success: A review of research. *Management Science*, 30, 586–603.
- Jarvenpaa, S. L., Rao, V. S., & Huber, G. P. (1988). Computer support for meetings of groups working on unstructured problems: A field experiment. *MIS Quarterly*, 12, 645–666.

- John, P. A. (1988). The ergonomics of computer aided design within advanced manufacturing technology. *Applied Ergonomics*, 19, 40–48.
- Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9, 256–278.
- Kaiser, K. M., & King, W. R. (1982). The manager-analyst interface in systems development. *MIS Quarterly*, 6(1), 49–59.
- Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). New York: Plenum.
- King, W. R., & Cleland, D. I. (1975). The design of management information systems: An information analysis approach. *Management Science*, 22, 286–297.
- Kinston, W. (1986). Purposes and the translation of values into action. *Systems Research*, 3, 147–160.
- Konsynski, B. R., Kottemann, J. E., Nunamaker, J. F., Jr., & Stott, J. W. (1984–1985). PLEXSYS-84: An integrated development environment for information systems. *Journal of Management Information Systems*, 1, 64–104.
- Kottemann, J. E., & Konsynski, B. R. (1984). Dynamic metasystems for information systems development. In *Proceedings of the 5th International Conference on Information Systems* (pp. 187–204).
- Koubek, R. J., Salvendy, G., Dunsmore, H. E., & LeBold, W. K. (1989). Cognitive issues in the process of software development: Review and reappraisal. *International Journal of Man-machine Studies*, 30, 171–191.
- Kozar, K. A., & Mahlum, J. M. (1987). A user generated information system: An innovative development approach. *MIS Quarterly*, 11, 163–171.
- Kramer, J., & Ng, K. (1988). Animation of requirements specifications. *Software — Practice and Experience*, 18, 749–774.
- Kull, D. (1987). Software development: The consensus approach. *Computer & Communications Decisions*, 19(11), 63–64, 66–67.
- Land, F. F. (1982). Participation in the information systems field. *The Computer Journal*, 25, 283–285.
- Lansdown, J. (1987). The creative aspects of CAD: A possible approach. *Design Studies*, 8, 76–81.

- Leonard-Barton, D. (1987). The case for integrative innovation: An expert system at Digital. *Sloan Management Review*, 29(1), 7-19.
- Loucopoulos, P., Black, W. J., Sutcliffe, A. G., & Layzell, P. J. (1987). Towards a unified view of system development methods. *International Journal of Information Management*, 7, 205-218.
- Malhotra, A., Thomas, J. C., Carroll, J. M., & Miller, L. A. (1980). Cognitive processes in design. *International Journal of Man-machine Studies*, 12, 119-140.
- Meister, D. (1987). A cognitive theory of design and requirements for a behavioral design aid. In W. B. Rouse & K. R. Boff (Eds.), *System design: Behavioral perspectives on designers, tools, and organizations* (pp. 229-244). New York: North-Holland.
- Mumford, E. (1979). Consensus systems design: An evaluation of this approach. In N. Szyperski & E. Grochla (Eds.), *Design and implementation of computer-based information systems* (pp. 221-230). Alphen aan den Rijn, The Netherlands: Sijthoff & Noordhoff.
- Nadler, G. (1981). *The planning and design approach*. New York: Wiley.
- PLEXSYS user guide*. (1989). Tucson: The University of Arizona, Department of Management Information Systems.
- Rasdorf, W. J. (1985). Perspectives [sic] on knowledge in engineering design. In R. Raghavan & S. M. Rohde (Eds.), *Computers in Engineering 1985: Proceedings of the 1985 ASME International Computers in Engineering Conference and Exhibition: Vol. 2* (pp. 249-253). New York: The American Society of Mechanical Engineers.
- Rouse, W. B. (1986). On the value of information in system design: A framework for understanding and aiding designers. *Information Processing & Management*, 22, 217-228.
- Rouse, W. B., & Boff, K. R. (1987). Designers, tools, and environments: State of knowledge, unresolved issues, and potential directions. In W. B. Rouse & K. R. Boff (Eds.), *System design: Behavioral perspectives on designers, tools, and organizations* (pp. 43-63). New York: North-Holland.
- Rouse, W. B., & Cody, W. J. (1988). On the design of man-machine systems: Principles, practices and prospects. *Automatica*, 24, 227-238.

- Rowen, R. B. (1990). Software project management under incomplete and ambiguous specifications. *IEEE Transactions on Engineering Management*, 37, 10–21.
- Shamlin, C. (1989). *A user's guide for defining software requirements: The other side of software* (2nd ed.). Wellesley, MA: QED Information Sciences.
- Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106, 467–482.
- Soloway, E. (1986). What to do next: Meeting the challenge of programming-in-the-large. In E. Soloway & S. Iyengar (Eds.), *Empirical studies of programmers* (pp. 263–268). Norwood, NJ: Ablex.
- Spectrum staff. (1990). The case for CASE tools. *IEEE Spectrum*, 27(11), 78–81.
- Srinivasan, A., & Davis, J. G. (1987). A reassessment of implementation process models. *Interfaces*, 17(3), 64–71.
- Swanson, E. B. (1976). The dimensions of maintenance. In *Proceedings of the 2nd International Conference on Software Engineering* (pp. 492–497).
- Swartout, W., & Balzer, R. (1982). On the inevitable intertwining of specification and implementation. *Communications of the ACM*, 25, 438–440.
- Tanimoto, S. L. (1987). *The elements of artificial intelligence: An introduction using LISP*. Rockville, MD: Computer Science Press.
- Teichroew, D., & Hershey, E. A., III. (1977). PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems. *IEEE Transactions on Software Engineering*, SE-3, 41–48.
- Thomas, J. C., & Carroll, J. M. (1979). The psychological study of design. *Design Studies*, 1, 5–11.
- Thomas, J. C., & Carroll, J. M. (1981). Human factors in communication. *IBM Systems Journal*, 20, 237–263.
- Ventana Corporation. (1990). *GroupSystems Version 4.0*. Tucson, AZ: Author.
- Vitalari, N. P., & Dickson, G. W. (1983). Problem solving for effective systems analysis: An experimental exploration. *Communications of the ACM*, 26, 948–956.

- Vogel, D. R., Nunamaker, J. F., Jr., George, J. F., & Dennis, A. R. (1988). Group decision support systems: Evolution and status at The University of Arizona. In R. M. Lee, A. M. McCosh, & P. Migliarese (Eds.), *Organizational decision support systems* (pp. 287–304). Amsterdam: North-Holland.
- Volkema, R. J. (1988). Problem statements in managerial problem solving. *Socio-economic Planning Sciences*, 22, 213–220.
- Voss, J. F., & Post, T. A. (1988). On the solving of ill-structured problems. In M. T. H. Chi, R. Glaser, & M. J. Farr (Eds.), *The Nature of Expertise* (pp. 261–285). Hillsdale, NJ: Erlbaum.
- Walz, D. B., Elam, J. J., Krasner, H., & Curtis, B. (1987). A methodology for studying software design teams: An investigation of conflict behaviors in the requirements definition phase. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), *Empirical studies of programmers: Second workshop* (pp. 83–99). Norwood, NJ: Ablex.
- Ward, T. (1987). Design archetypes from group processes. *Design Studies*, 8, 157–169.
- Warfield, J. N. (1973). Intent structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3, 133–140.
- Webster, D. E. (1988). Mapping the design information representation terrain. *Computer*, 21(12), 8–23.
- White, K. B., & Leifer, R. (1986). Information systems development success: Perspectives from project team participants. *MIS Quarterly*, 10, 215–223.
- Wulz, F. (1986). The concept of participation. *Design Studies*, 7, 153–162.
- Yadav, S. B. (1983). Determining an organization's information requirements: A state of the art survey. *Data Base*, 14(3), 3–20.
- Zmud, R. W. (1980). Management of large software development effort. *MIS Quarterly*, 4(2), 45–55.