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.

UMI

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

TRACKING CONSENSUS IN PRODUCT DEVELOPMENT TEAMS

by

Jeffrey A. Morrow

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

University of Washington

1998

Approved by _____
Chairperson of Supervisory Committee

Program Authorized to Offer Degree: Industrial Engineering Program

Date August 8, 1998

Doctoral Dissertation

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature _____

Date August 8, 1998

University of Washington

Abstract

TRACKING CONSENSUS IN PRODUCT DEVELOPMENT TEAMS

by Jeffrey A. Morrow

Chairperson of the Supervisory Committee
Professor Joseph A. Heim
Industrial Engineering Program
Department of Mechanical Engineering

Modern concurrent engineering requires that the perspectives of the stakeholders in the product realization process be harmonized as early as possible in that process. We consider this harmonization to consist largely of reaching consensus about stakeholder perspectives and our research seeks to find ways to measure and characterize the states of consensus of product development teams. We experimented with various methods of cognitive anthropology, namely pile sorts, triads tests, ranking, and magnitude scaling, to elicit data and to test respondents' compliance. To simplify data elicitation, we wrote software in Microsoft Visual Basic to emulate these methods on personal computers. We used Cultural Consensus Theory and graphical methods to analyze results, concluding that it is possible and feasible to track consensus states in product development teams. We make suggestions for further work in the methods of cognitive anthropology and to extend the software emulations.

Key words: Consensus, product development, cultural consensus theory, pile sort, magnitude scaling

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Affinity Diagram. A method for collectively eliciting elements of a domain of interest from a group of people and collectively clustering those elements into a conceptual hierarchy. Affinity diagrams have roots in cognitive anthropology and have been extensively developed by Jiro Kawakita. For instruction in making an affinity diagram, please see Appendix B.

Cluster Analysis. A category of analytic methods that, by various means, typically collect scatter-plotted data into groups based on relative similarity. Often the clusters are taken as evidence of hierarchy and used to generate a taxonomy.

Cognitive Anthropology. A genre of anthropology which established itself in the late 1960s and attends to study of how humans think about salient aspects of cultures in which they are members. Cognitive anthropology includes the subcategories of ethnographic semantics, semantic theory, ethnoscience, scripts and schemata, folk models, decision models, goal structures and motivational systems, narrative grammars, discourse semantics, and content analysis.

Cognitive Domain. Some group or cluster of elements of thinking organized in a self-evident way in the mind of a person or in the minds of a group of people. A cognitive domain may be as evident as kinship or as vague as "books that are important." Also called schemas or frames.

Compliance. The amount of response achieved on administration of a data elicitation instrument compared to that sought. High compliance respondents complete all the administrations faithfully.

Concurrent Engineering. The attempt to concurrently design products and the processes that produce them. Also called simultaneous engineering. Contrast this with serial engineering wherein the product is completely designed and then that design is presented to those who design the production methods.

Consensus. The sharing of perspective regarding a cognitive domain among a group of people.

Dissensus. The lack of sharing of perspective regarding a cognitive domain.

Freelist. A collection of cognitive domain elements generated by a member of a cohort that shares the cognitive domain. Most frequently a freelist is composed of words but on occasion consists of physical objects.

Graphical User Interface. The means by which a user operates a computer, manipulating graphical elements in addition to text. Perhaps the earliest example of a graphical user interface is the SmallTalk interface developed by the Xerox Palo Alto Research Center.

Multidimensional Scaling. The use of a number of computational techniques to project into a low-dimensioned space (most typically a plane, or 2D space) points from a higher-dimensioned space in such a fashion that the distances between pairs of points remain proportional. Such projection is an optimization problem, the attempt being to minimize some objective function characterizing the proportionality errors.

Pile Sort. The elicitation of data regarding similarity between elements of a cognitive domain by using a variety of protocols in which respondents are asked to cluster domain elements on the basis of "belonging together."

Proximity Matrix. The matrix of pairwise similarities between domain elements. In a proximity matrix, a larger number indicates greater similarity between two elements. Contrast this with a distance matrix, in which a larger number indicates less similarity between elements. Also called a similarity matrix.

Quality Function Deployment. A reliable method for identifying, amplifying, clarifying, harmonizing, translating, and sharing the preferences of customers among those who are developing products for the customers, in order that the resultant product provides most customer satisfaction. Quality Function Deployment is a complex, painstaking method that has achieved best application in certain world-class Japanese companies.

Singular Value Decomposition. Use of any of a number of computational techniques (e.g., residuals minimization) to extract the eigenstructure from correlated data. More colloquially, an attempt to explain the sources of variation in a data set in terms of a single variable.

Thought World. A cognitive domain shared by members of a functional subset of an enterprise that develops products.

Triads Test. The serial presentation of all or some combinations of elements of a cognitive domain, three at a time, and the request that the respondent select the one that "belongs least" or equivalent. A triads test produces pairwise similarity data.

PREFACE

In this investigation, we place few restrictions on the nature of products developed: almost any collective effort to analyze market demand and synthesize a profitable response would be accommodated. To do so, an appropriate (relatively high) level of abstraction of the product development (PD) process must be assumed. We take the perspective that the work of PD teams can be split between design accomplishments and execution accomplishments.

With regard to design accomplishments, the PD team must solve a stream of highly variegated constrained design problems during the PD cycle (e.g., negotiating the PD schedule and its resourcing, finding a concept design, debugging the shipping process, to name only a few). Yet the cleverness of any collective design does not suffice that it succeeds: the development team must execute its clever designs with fidelity and energy. A bit of good fortune may be necessary, too.

When designing solutions for mathematical word problems as students, we all experience the special kind of breakthrough that occurs when shifting to a more subtle (now obvious) perspective after having struggled with an inadequate one. In the same vein, Albert Einstein is said to have claimed that "Perspective is worth 20 IQ points." More recently, Page has shown that for a certain class of search problems, any paramount solution that can be reached by exhaustive application of a search strategy can be found directly by correct rotation of the search space. (Page 1995). "Correct rotation of the search space" may be considered equivalent to adopting a different perspective.

There is also a large and growing literature regarding the importance of accommodating the perspectives of all the stakeholders in the PD process. It has become part of the canon of concurrent engineering that some of the benefits of product development done by teams

flows from early accommodation of the diverse perspectives held in relevant enterprise functions. To these ends, it is further held that PD teams ought to be comprised of representatives of all of the important development domains and that these members have easy communication, perhaps to the extent of being collocated.

More recently, some researchers have found that differences in departmental "thought worlds" are also related to PD team dysfunction: process disruption, trust and respect challenges, difficulties in achieving consensus for action, etc. (Pelled and Adler 1994; Dougherty, 1992) Thus, while it seems that concurrent or simultaneous engineering clearly brings diversity of perspective to product development (if only by lumping representatives from relevant functional organizations together as a team), this diversity of perspective does not seem to represent an unalloyed good when thought worlds collide.

It seems likely that successful PD requires balance between diversity of perspective and ability to execute in an aligned and energetic fashion. Clausing suggests that the prime benefit of expert use of Quality Function Deployment (QFD), a method for PD teams to ensure that customer preference guides all the important choices in development, comes in the strong consensus generated. From Clausing's perspective, strong consensus yields coordinated and timely PD team execution. (Clausing, 1994) Of course, the other prime benefit of expert QFD use is that it sifts and harmonizes great diversity of perspective very reliably.

What we seek, then, is to characterize the states of consensus of the PD team throughout the PD cycle. More explicitly, we define achieving consensus as reduction in "diversity of perspective" over time. While we expect that PD teams starting with greater diversity of relevant perspective regarding product design and withal having greater ability to achieve consensus despite the initial diversity of perspective, will more reliably develop good products, gauging PD success is such a daunting task that in this research we attempt only to measure consensus change.

From the literature of cognitive anthropology, we find theory and methods that permit characterization of, among other things, locus and dispersion of cognitive mental models held by individuals in a group or culture. (D'Andrade 1995) We take advantage of Cultural Consensus Theory and the methods of the pile sort, magnitude scaling, and ranking. Each of these methods can be very researcher-labor intensive and typically requires researcher physical presence for administration (with attendant problems in scheduling with respondents). This drawback only worsens when, as we require, locus and dispersion of perspective must be tracked over time, thus necessitating multiple reassessments. As classically administered, these methods are probably too burdensome to use for investigating product development diachronically or for teams that are geographically dispersed. Our research depends on novel versions of these tools of cognitive anthropology that we have implemented as PC software and that go some distance to mitigating application burden of the tools.

Our contributions are to evaluate methods for measuring putatively important aspects of product development, to build, debug, and evaluate novel, computer-mediated versions of standard methods of cognitive anthropology, and to propose method extensions that the computer versions allow.

ACKNOWLEDGMENTS

The author wishes to acknowledge the key contributions to this research made by Professor Joseph A. Heim, dissertation committee chair and Professor Kailash Kapur, the director of the Industrial Engineering Program. It is safe to say that this work would not have been completed without their support and criticism.

The author's other dissertation committee members also deserve special acknowledgment. Professor Robert Smith consistently provided many highly useful suggestions and much of the credit for any intellectual rigor belongs to him. Professor Marc Miller unfailingly gave authoritative guidance in navigating anthropology and was a great moral support, often at just the right time. If Professor Dale Calkins had not facilitated access to the SAE Formula Car '97 team, a large amount of relevant data and administration experience would have been unavailable. Professor Michael Pilat, the Graduate School Representative, helped keep the examinations running smoothly and also gave timely encouragement.

Professor James Solberg of Purdue University provided crucial funding and access to an important population of respondents. The month the author spent in West Lafayette as a result was essential for the research.

The author also wishes to acknowledge the support, insight, connections, and encouragement in anthropology given by Dr. Devon Brewer, Professor H. Russell Bernard, and Professor Eugene Hunn.

The author's employer deserves acknowledgment for his forebearance and the rare flexibility in work arrangements with which he accommodated the research.

Finally, the author acknowledges with gratitude the freely given help given by his respondents. Without their voluntary kindness, there would be no research.

DEDICATION

To Red, for her love, respect, sacrifice, and goading.

Without you, nothing.

INTRODUCTION AND PROBLEM STATEMENT

WHAT WE INVESTIGATED

This research has exploration for its rationale at least as much as it seeks scientific proof. It is motivated by a conjecture of Don Clausing's that the key benefit of expert Product Development (PD) team use of Quality Function Deployment (QFD) comes in the strong consensus generated. (Clausing 1994) Clausing and others have previously advocated QFD use primarily as a means to ensure that customer preference guides decision making in a rigorous way throughout the entire product realization process. (Hauser and Clausing 1988) To fully test the **Clausing Conjecture** we would need to evaluate PD project outcomes and these are the results of many confounded factors. We would need to measure differences and changes in motivation, expertise, resourcing, schedule fidelity, and product market value, at least, in addition to consensus. That being beyond the scope of this research project, we have chosen to take a first step of finding means of tracking difference and change in consensus in practicing product development teams. For such methods to work they need to burden respondents minimally and acceptably. The methods and analysis should have face validity and pass relevant academic norm acceptance criteria. To select the better methods, we need to investigate a variety of data elicitation methods using a variety of presentation media. Different analysis methods should be tried. It would be well if our respondents had relevant characteristics in common with members of product development teams. The remainder of this chapter consists of subsections that discuss the above issues. On the next page is a schematic depicting the research strategy, annotated with relevant section titles from the dissertation.

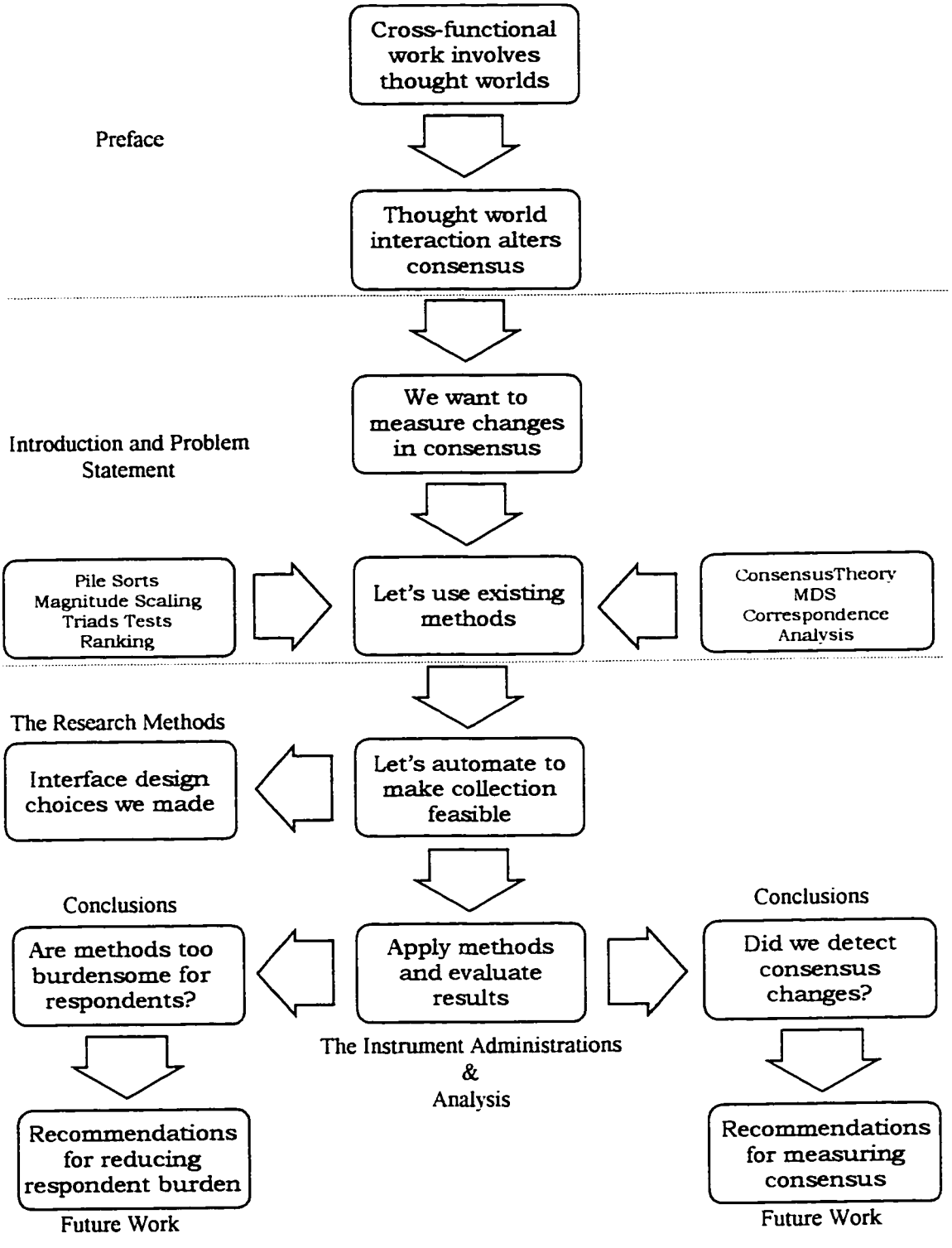


Figure 1: A Schematic of Research Strategy and Dissertation Structure

How much burden will people put up with?

We chose method and administration variety so that a range of burden was imposed. We chose "average time to complete" as a good surrogate for burden and investigated tasks that could be done in one minute and some that might take 30 minutes, assuming that a 15 minute long administration done weekly well enough estimates what PD practitioners would find acceptable. Our research methods timed how long most administrations took.

Which presentations seem most burdensome?

We elicited data with a variety of methods using a variety of presentations. Specifically, we used freelisting, pile sorting, magnitude scaling, ranking, and triads tests. Methods were administered using physical card decks, computer images of card decks, and paper and pencil tasks.

How can a Graphical User Interface (GUI) help? Hurt?

We proposed a variety of "click-drag-drop" syntaxes for piling and scaling tasks and evaluated differing numbers and layouts of cards and differing card sizes. We attempted to elicit semantic (meaning) data by offering appropriate text boxes for respondents to comment in.

What sort of mistake proofing can we do?

In physical administration of these instruments, researcher consistency in presentation and standard data elicitation is important. In our software embodiment, we experimented to find which choices and actions ought be blocked, where to have error trapping routines that fail safe, and how to have kinesthetics that feel natural to respondents.

CAN WE DISCRIMINATE PERSPECTIVE VARIANCE?

We chose cognitive domains of relative salience for respondents. Presumably they responded non-randomly and based their response on a personal schema that organized the cognitive domain for them. Some administrations presented minimal cognitive armature for the respondent and others presented dimensions for scaling that many product development practitioners deem relevant. We hoped to test if a variety of domain element sorts was unlikely to have been done at random.

HOW DOES CULTURAL CONSENSUS THEORY APPLY?

Cultural Consensus Theory is an established method in cognitive anthropology. Predicated on certain assumptions about a given cognitive domain not being falsified, it asserts that domain element similarity data may be analyzed by a protocol that yields valid inferences about amount of domain knowledge any respondent has, correlations between respondents in terms of agreement about the domain, and prediction of the most correct arrangement of the cognitive domain. We applied the analysis pertinent to Cultural Consensus Theory and drew appropriate conclusions where warranted.

CAN MULTIDIMENSIONAL SCALING OFFER INSIGHTS?

Multidimensional Scaling (MDS) attempts to fit into (typically) two dimensions a map of n domain elements in which every pair of elements is separated on the plane by a distance proportional to the perceived similarity between them in the n-1 space of pairwise comparisons. Here, more similarity implies a shorter distance between a pair of domain elements. When this can be done with an acceptable amount of error, making inferences about the nature of each axes gradients is considered meaningful. It is best, of course, to test these inferences explicitly after detecting evidence for them. (Borgatti 1992) The clustering of elements in the plane may be considered meaningful, too. We performed MDS on data that permitted it and evaluated the results.

How might information theory aid analysis?

When few data are elicited, as occurs in minimizing burden, less information can be gleaned. When response is less structured, as in pile sorts, respondents take a larger variety of perspectives. When data are few and native perspective variance is high, analysis can stand on shaky ground: too many views too abstractly characterized will drop the signal to noise ratio. In 1972, Michael Burton (Burton 1972) proposed an information theoretic transform that might be done on free sorted element pairs to compensate for respondent tendencies to sort domains into less abstract, smaller clusters ("splitters") or to combine most cards into more abstract, inclusive clusters ("lumpers"). We have applied his transform and evaluated the results.

Plan of attack

Instrument SAE Formula Car '97

1. Present a physical card importance scaling in a domain of low elaboration (few elements).
2. Present a computer-mediated pile sort with a new domain also of low elaboration; use the "full Boster" protocol and make rationale entry optional.
3. Present a computer-mediated triads test with the same domain as the previous pile sort.
4. Present a computer-mediated free sort pile sort with a new domain of high elaboration; make rationale entry optional.
5. Present a computer-mediated free sort readministration of the low-elaboration domain pile sort; make rationale entry optional.
6. Present a pencil and paper importance scaling with original domain of low elaboration.

INSTRUMENT THE SUMMER 1997 SPECIAL PROJECT

1. For each of five teams, present every team member with a computer-mediated pile sort of that team's "project activities" domain; administer to each team member at beginning, middle, and end of project.

2. For each of five teams, present every team member with a computer-mediated importance scaling of that team's domain; administer three times with the pile sorts.

3. For each of five teams, present every team member with a computer-mediated difficulty scaling of that team's domain; administer three times with the pile sorts.

INSTRUMENT ASC DERIVATIVE PRODUCT DEVELOPMENT

1. Present every team member with a computer-mediated pile sort of the team's "project activities" domain; administer to each team member at beginning and end of project.

2. Present every team member with a computer-mediated importance scaling of the team's "project activities" domain; administer twice with the pile sorts.

3. Present every team member with a computer-mediated difficulty scaling of the team's "project activities" domain; administer twice with the pile sorts.

4. Display intermediate analyses to team members as performed.

CHAPTER 1: USING ANTHROPOLOGY IN PRODUCT DEVELOPMENT

INTRODUCTION

If only because of the apparent disparity between the domains of anthropology and product development, some words of definition and motivation are indicated at the outset of a paper linking the two. First, let us consider anthropology. Anthropology is one of the social sciences. It evolved as a distinct discipline in the late nineteenth century, distinguishing itself in part from the other social sciences by an initial focus on non-Western human culture. (Colby 1996:209-211) Possibly because of these roots, anthropology, of all the social science disciplines, makes the greatest effort to stand outside any context, to strip away the assumptions of the researcher's own culture. Anthropology's more intimate methods, particularly ethnography, also tend to differentiate it from other social sciences' somewhat distanced approaches. For example, after an ethnographer crosses borders into another cultural context, she participates in and observes directly that culture rather than standing apart from it, as a sociologist might in simply administering surveys. This direct experience of fieldwork, often enough done in awkward, uncomfortable circumstances for extended periods, has become almost a professional rite of passage for some anthropologists. The anthropologist goes to see the real thing, feel the real thing, record meticulously and then carefully construes meaning in her experience. Having found meaning in the foreign context, the anthropologist forges links (sometimes implicitly) between her native culture and the foreign one, and in the process reveals universality.

Beginning in the late 1950s a branch of anthropology grew that initially concerned itself with heightened attention to elicitation of cultural data. The researchers' intent was to minimize blind impression of their own cultures' thinking in field data elicitation. For example, at the time a common ethnographic activity involved the eliciting of names of

things in cultures. Typical practice had the ethnographer indicate an object, ask "What is this called?" and associate the local word with the appropriate word from the ethnographer's culture. On the face of it, this may seem straightforward and uncontaminating, but subtle ethnographers saw the fallacies and from their insights came studies that showed that how a culture organizes *thinking* affects thinking about *things*. As Charles O. Frake, a cognitive anthropology pioneer, put it

> "If, however, instead of 'getting words for things,' we redefine the task as one of finding the 'things' that go with the words, the eliciting of terminologies acquires a more general interest. In actuality not even the most concrete, objectively apparent physical object can be identified apart from some culturally defined system of concepts." (Frake 1969)

In 1969 *Cognitive Anthropology*, a book collecting papers written in the preceding two decades, was published, naming this new field and establishing it as a genre of anthropology. (Tyler 1969) In the book, many of the research themes of the subsequent decade, folk-biological taxonomy, color perception, kinship terminology, cultural data elicitation, among others, were developed. The 1970s saw application of new tools to these themes, principal among them multidimensional scaling and cluster analysis, both of which are useful for drawing out of data its underlying structure. (Romney, *et al.* 1972). In the 1980s Cultural Consensus Theory was developed and elaborated by cognitive anthropologists, partly in furtherance of doing efficient ethnoscience while minimizing the assumption of cultural knowledge on the part of the researcher. Because Cultural Consensus Theory is at the core of our research, we postpone description until the fourth section of the paper in which we describe it and our work.

Figure 1 (Colby 1996:210) presents a schematic view of anthropology. For us, while it might be interesting to use archeology to investigate Paleolithic product development, cognitive anthropology and social anthropology were more relevant. Within cognitive anthropology our attention has been to ethnoscience, consensus theory and discourse

analysis. Within social anthropology (not broken out), our attention has been to ethnography.

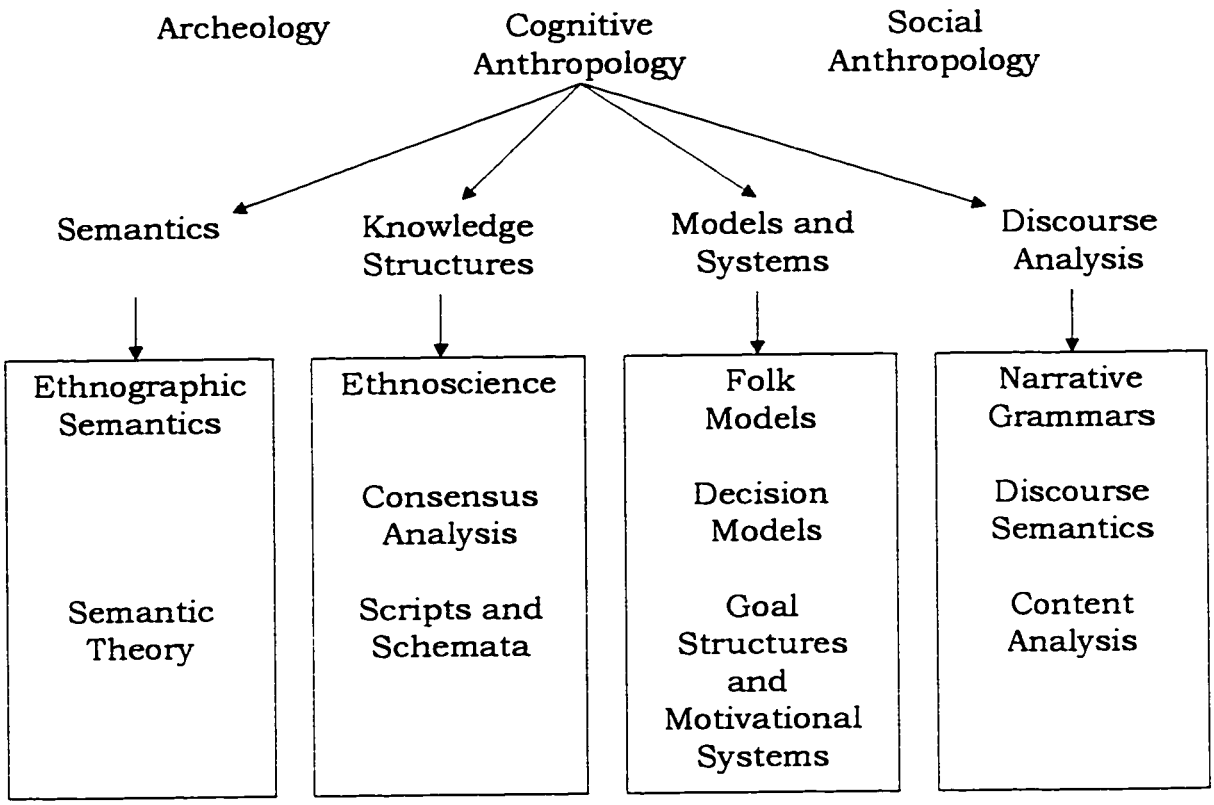


Figure 2: Situating Cognitive Anthropology within Anthropology (Colby 1996:210)

Turning now to product development (PD), we note that there is a large and growing literature, spanning economics, engineering, management science, and management, that addresses the topic. Brown and Eisenhardt provide an excellent review of this literature and we will not attempt to cover that ground here. (Brown and Eisenhardt 1995) Most generally almost any collective effort to understand market demand and synthesize a profitable commercial response can be considered product development. Our interest has been in regarding the importance of accommodating the perspectives of all the stakeholders in the PD process. It has become part of the canon of modern engineering

that some of the benefits of product development done by teams flows from early accommodation of the diverse perspectives held in relevant functional departments. To these ends, it is held that PD teams ought to be comprised of representatives of all of the important development functions and that these members must have easy communication, perhaps to the extent of being physically collocated. (Allen 1977)

Some researchers have found that these perspective differences in departmental "thought worlds" are also related to PD team dysfunction: process disruption, trust and respect challenges, difficulties in achieving consensus for action, etc. (Pelled 1994; Dougherty 1992) Thus, while it seems that modern concurrent engineering clearly brings diversity of perspective to product development (if only by lumping representatives from relevant functional departments together as a team), such diversity of perspective does not always represent an unalloyed good when thought worlds collide.

What we might seek, then, is to characterize the states of perspective consensus of the PD team throughout the PD cycle. More explicitly, we will define achieving consensus as reduction in diversity of perspective over time. More speculatively, we might anticipate that PD teams beginning with greater diversity of relevant perspective regarding any design problem, and withal having greater ability to achieve consensus despite initial diversity of perspective, would more reliably develop good products.

This chapter primarily surveys application of anthropology to two aspects of product development: understanding the nature of market demand, and understanding the culture of product developers. Neither literature is particularly large though both have grown in recent years. Indeed, as recently as 1989, Leong found in a citation analysis of the *Journal of Consumer Research* that only 1.5% of articles published referenced anthropological methods. (Leong 1989) Yet, in a recent database search of Association for Computing Machinery (ACM) publications, a simple query for articles containing both "product development" and "ethnography" returned over 100 references and those 100 references pointed to over 600 related ACM references. In certain circumstances, product

developer needs constitute the market, as in some research in Computer Supported Collaborative Work (CSCW) where the market opportunity lies in finding computer-mediated methods for product development teams. The first section discusses earlier applications of cognitive anthropology to understanding the ways that customers think about products like coffee, pain relievers, and breakfast cereal. The second section describes current employment of anthropology in support of improving Computer Supported Collaborative Work (CSCW) and for other market research purposes. CSCW application has predominantly been for the purposes of finding ways to facilitate group work by employing computer systems, and to coordinate the dialectic that exists between support of current work practice within a community and the alteration of practice caused by introduction of new community tools. The other applications of ethnography to market research are much more diverse. The third section reviews examples of ethnography, a subdiscipline of social anthropology, used within the borders of product developer communities to understand cultures of product development. The fourth section outlines our approach to using computer-mediated versions of cognitive anthropology methods to characterize how product developer team consensus/dissensus changes across the cycle of development.

Figure 2 shows where applications of the various methods of anthropology discussed in this paper fit in an idealized cycle of product development. Beginning closest to the customer, on the left, we find use of ethnography and ethnoscience to understand consumer thinking. Slightly nearer the development of products (but actually spread throughout the link between customers and product developers) is Computer Supported Collaborative Work. Situated within the product developer domain, at the top of the diagram, is Product Development Project Ethnography - typically the participant-observer.

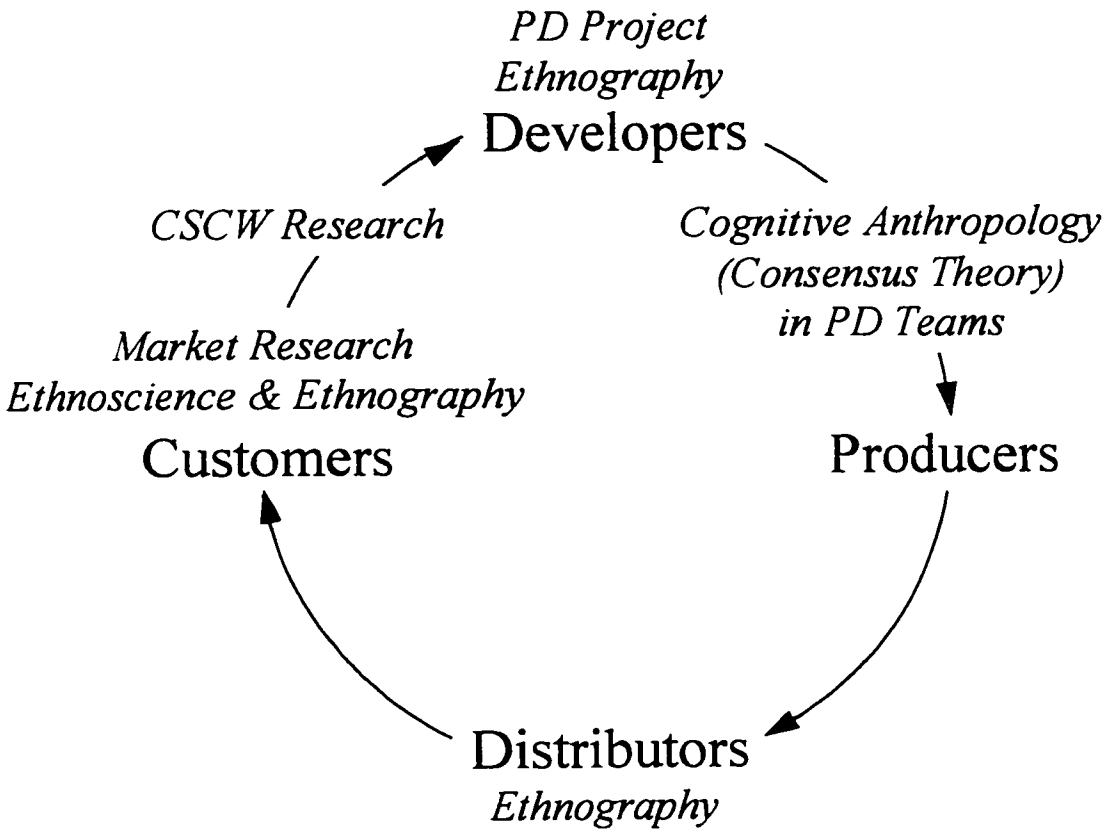


Figure 3: Situating Various Methods of
Anthropology in Product Development

study of engineers developing products on a project basis. On the right, spanning the distance between product development and production (to symbolize a modern trend toward concurrent or simultaneous product/process/production development) is our application of cognitive anthropology. Finally, it is perhaps interesting to note that we found only two published applications of anthropology that could conceivably fit in the Distributor portion of the product development cycle: one, an ethnography of corporate purchasing agent culture. (Wilson 1996) and the other, an ethnography of West African export marketing transactions to recommend strategies for improvements in channel performance (Arnould and Iddal 1992) However it seems likely that some considerable (and possibly scientific) effort has been expended to understand this domain, leaving open

the opportunity for research into things like the culture of Wal-Mart's fabled logistics system and the ethnography of distribution channel management

ETHNOSCIENCE: COGNITIVE ANTHROPOLOGY IN MARKET RESEARCH

Application of methods of cognitive anthropology to product development have been myriad, if one defines cognitive anthropology loosely enough and doesn't mind stepping on other social sciences' sensibilities, because commercial practitioners can be avid if not always discriminating users of social science methods. The commercial practitioners' results are mostly fugitive, however, because they haven't the time or interest to publish, especially if such might reveal proprietary information. (Mitchell 1998) Carmone and Green, in a 1972 paper, discuss use of similarity assessment of advertising images to establish whether nominal experts really know more about the nature of the images than randomly chosen graduate students. (Green 1972) Here the centrality of certain informant's similarity judgments is used to identify them as expert. Finally, Green and Carmone allude to, but do not describe, application of these methods to soft drinks, breakfast foods, graduate business schools, automobiles, cold cereals, common stocks, TV shows, industrial vendors, and ladies underwear. (Green 1972)

In the same volume, Stefflre (a principal within a consulting firm as well as a professor) comments on a decade of research spent in the hopes of finding ways to collect data that will permit prediction of behavior - much of that behavior having commercial ramifications. Although his paper focuses on methods, a safer topic than "results in commercial application," we find in the paper examples of application of cognitive anthropology methods to understanding how Americans view taking over-the-counter pain medication (Bufferin was the client, apparently), how Americans think about when and what snacking is appropriate, how Americans think about similarities between cigarette brands, and how Americans imagine the Peace Corps experience to be. Perhaps his most dramatic example is of the construction of both a physical product (freeze-dried

coffee) and the image of that coffee (Taster's Choice). For his clients, Stefflre mapped the cognition of Americans regarding coffee, discovered a region in that map that had no matching product, helped the producer define the organoleptics of the coffee to match that region, provided the underlying themes the producer would need to position the product in the market, and rather successfully predicted the market share that Taster's Choice would take. Undescribed, but mentioned, are:

> "(T)hree other (projects) of the same type" and "five experimental cases in which we (a) found out how people described a product under development (four cases) or just introduced it into a test market (one case), then (b) tested the product's description in a 1500-person sample, and finally (c) obtained data on the product's performance in the regional or national market in which we had tested its description." (Stefflre 1972)

COMPUTER SUPPORTED COLLABORATIVE WORK AND MARKET RESEARCH ETHNOGRAPHY

As a distinct community of research, Computer Supported Collaborative Work (CSCW) is scarcely a decade old but has high visibility among computer-system attentive academics and, increasingly, the attention of leading edge consulting and developing firms involved with computer and computer-dependent industry. In a description of a 1996 course offering, "The Ethnography of Information Systems," two leading CSCW academics, Geoffrey C. Bowker and Susan Leigh Star provide a useful brief overview of the field. According to them,

> "(E)arly work focused on philosophical and epistemological divergence, with a critical edge and somewhat arms' length relationships (Suchman 1987). Since the mid -1980s, especially in Britain and Scandinavia and increasingly in the US, full-fledged partnerships have grown. All have in common the goal of analyzing the contingencies of information-based work practice as situated in particular times and places, and using that analysis to inform user-sensitive information systems design." (Bowker 1997)

Put in other words, CSCW started out by differentiating itself from conventional computer system human interface design by adopting a critical and distanced stance, the establishment of such a stance being then very salient in high academic anthropology. As

it matured it has become more commercially congenial, intentionally less theoretical, and deeply focused on the actual, situation-dependent ("contingent") real world of people using computers in collaborative ways. Suchman's form of CSCW falls primarily in the Discourse Analysis category of Figure 2; others most frequently term themselves ethnographers, a subdiscipline of Social Anthropology.

According to Bowker and Leigh Star, some specific examples of CSCW research are:

- use of ethnomethodology for determining requirements for air traffic control
- use of discourse and workplace analysis to study communication and design of organizational communications systems
- use of video and conversation analysis to generate system requirements in finance and medicine and for medical data transfer protocols
- study of workplace conversation and interaction to improve safety-critical systems for controlling the London underground
- exploration of aspects of so-called ubiquitous computing with smart badges
- ethnographic study of nursing and surgical work
- studies of the visual practices of CAD engineers
- production of medical expert systems and other Artificial Intelligence investigations
- use of ethnography for development of computer systems for government entities

Under the general heading of CSCW is a large subcategory called Participatory Design that began in Scandinavia when national legislation mandated workplace assessment of new technology. In Europe, where union-management-government cooperation runs more deeply than in the U.S., support for structured effort to harmonize conflicting interests in product realization has fostered strong interdisciplinary study of workplaces.

Examples of this sort of CSCW include computer systems for graphic design, nursing support, and banking. (Bowker 1997) Despite CSCW's support and interest in harmonizing relations between the stakeholders in organizations, however, Participatory Design in European projects has run afoul of power imbalances among the stakeholders as Wagner and Gaertner report in "Mapping Actors and Agendas: Political Frameworks of Systems Design and Participation." (Gaertner 1996) The complications of power relations cause problems in the United States as well: Suchman, Blomberg, and Trigg discuss such challenges that arose in their research to facilitate litigation support work practice at a Silicon Valley law firm with new computer systems. (Blomberg 1996)

While many authors in the CSCW vein strive to ground their research in actual work practice they also discourse about their topics in an often highly abstracted, sophisticated way. (See, for example, Suchman 1987) Most abstractly, it matters little whether collaborative work is for the design of legal briefs or graphical user interfaces; consequently, the theory of CSCW applies to the work of designers of the computer systems, too. Nevertheless, most CSCW research attends to users before producers. As a distinguished practitioner of consumer product development, Arnold Wasserman, comments in an invited response to a paper (Brown 1994) intended to stimulate dialog around the ideas of situated work and design among senior researchers at the Xerox Palo Alto Research Center (a locus of much CSCW work):

> "Brown and Duguid address superbly the issue of how the physical and social content and context of design intent should be founded. It seems just as urgent to address how well-founded design intent gets confounded as its content undergoes negotiation inside the physical, social, and (one must add) political context called the *designer*. The reductive designer is as problematic as the canonical artifact or the self-evident user." (Emphasis in the original) (Wasserman 1994)

Here Wasserman displays a perspective more typical, perhaps, of the earlier period of CSCW and makes a critical, distanced, philosophical point about the complexity of product development. His point is that thinking of design as springing full blown from the

mind of an autonomous inventor seriously distorts the reality of what is truly a collective and conflicted process. Wasserman goes on to claim:

> "...a new kind of designer has come into being whose medium of user research, product development, documentation, testing, and publication are all software. As canonical designers and producers combined, they embrace more of the total designer context than do traditional designers. And their rapidly iterated, media-consistent way of independently publishing artifacts has much to teach the physical and social direction of all design. But for the immediate future, most designers of both hardware and software products are destined to slug it out in a producer melee called the *designer*." (Emphasis in the original) (Wasserman 1994)

In software development then, the lines between designers, producers, and users blur. This natural condensation offers useful ways for developers of other types of products, showing the difficulties and benefits of tightening the design-build-distribute-utilize loop.

Turning now to applications of ethnography to understanding markets other than computer-mediated ones, we note a spate of recent articles in the popular business press (Heath 1997, Posner 1996, Anonymous 1996a, Anonymous 1996b, Rayner 1997, others) with focus on use of ethnography for marketing, most of these articles featuring firms that specialize in these approaches and appear to be successful in selling their services. For those seeking a very thorough review and discussion of ethnographic methods in market research, Arnould and Wallendorf's article in the *Journal of Marketing Research* is an excellent source. (Arnauld and Wallendorf 1994) In this article, the authors carefully detail the practice of ethnography in general and provide thoughtful examples of actual application to market research in American thinking, feeling, and behavior concerning the Thanksgiving holiday. In addition, the article contains numerous relevant references to market research, to ethnography, and to the intersection between them and cites many examples of practical application.

THE ETHNOGRAPHY OF DEVELOPMENT OF ENGINEERED PRODUCTS

In Figure 1 social anthropology is not broken down into subcategories as is cognitive anthropology. If it were, we would find within it the field of ethnography. As a genre of anthropology, ethnography has a rich, dues paying kind of place among the others. Ethnography typically has required practitioners to situate themselves as participant-observers in quite foreign (and often uncomfortable) places for a relatively long (on the order of months if not years) time. Rigorously done, these practitioners will keep detailed records of their experience while they strive to fit unobtrusively in the foreign culture by building trust and reciprocity in real time. Yet even that is not enough, for descriptive reportage alone fails to deliver the synthetic insights that bridge the foreign and familiar; proper ethnography draws the universal from two (or more) sets of apparently different ways to construe reality. In other words, the researcher must interpret the foreign in terms of the familiar, description alone not being enough. (Bernard 1994)

In 1961, David Marples published "The Decisions of Engineering Design" after doing his research in a manner we would characterize as respectable ethnography. (Marples 1961) Based on several years' work as a participant-observer researcher within what were then high technology industries in England, Marples described the physical, social, political, and economic aspects of two plant process designs. From the wealth of detail collected he drew graphs embedding a general theory of collaborative engineering problem solving in two, differing instantiations. Judiciously, discreetly, Marples also hinted at the politics and practice of the situations with neatly drawn anecdote.

Louis Bucciarelli explicitly advocates use of ethnography to understand engineering design and provides two studies of the design process within engineering firms in a 1988 paper. (Bucciarelli 1988) In particular he observes engineering design as a social process rather than primarily done by individuals in isolation - despite the common perception of engineers as asocial loners. As part of his attempt to generalize to universals from these cultures, he identifies three types of discourse within the designing process: constraining,

naming, and deciding. Working from field notes taken at meetings, he provides anecdotes to explain and buttress his arguments. A corollary to his construing of design as social process is that "different participants think about the work on the design in quite different ways." (Bucciarelli 1988) Below we return to this theme when discussing our own work.

Bucciarelli takes the ethnographic approach even farther in his 1994 book *Designing Engineers*. (Bucciarelli 1994) For this book, carefully acting as a participant-observer, he followed the realization of three products in three industries from beginning to end. Here again we find evidence of careful data collection, thoughtful attention to his status as other, realism regarding the power relations and politics of the firms involved, and illustrative anecdote.

More recently, Anne Donellen has written *Team Talk: The Power of Language in Team Dynamics*. (Donellen 1996) Donellen, a sociolinguist, takes an explicitly ethnographic approach to characterizing six important dimensions of team dynamics: team member identification, interdependence, power differentiation, social distance, conflict management techniques, and negotiation process. Her book provides samples of real conversational episodes from thorough field research observation of product development teams in four large U.S. corporations and a detailed analysis of each of the four teams' talk on each of the six dimensions.

In 1996, Sutton and Hargadon published "Brainstorming Groups in Context: Effectiveness in a Product Design Firm," which, despite its narrow titular focus, shows that the authors took an ethnographic approach to their research. (Sutton and Hargadon 1996) Both authors were experienced in the field as participants before becoming observers, and one worked at the firm studied for some time before and after the research. In treating the brainstorming topic, the authors address issues of status, power relations, commitment to projects, change management in firms, and group effectiveness.

While it cannot be deemed academic, much less scientific, *The Soul of a New Machine*, Tracy Kidder's book of a year in the life of a team of designers madly struggling to deliver a new computer and its operating system, also has the hallmarks of ethnography. (Kidder 1981) Kidder achieved successful immersion as a participant-observer over the long haul and made an inspired attempt to make sense of the foreign culture in familiar terms. This, too, he achieved if success in bringing the foreign into the familiar is evidenced in best seller status for a book about, among other things, the writing of compilers. Kidder shows that the universals of passion, commitment, pride, manipulation, and triumph were as much part of succeeding in the presumably formal world of minicomputer manufacture as they might be in mounting a Broadway show.

To conclude this section, let us acknowledge ethnographic truth can be told in surprising ways. After years as a literal participant-observer, Scott Adams has achieved fame and considerable fortune from publication of his wickedly insightful cartoon series *Dilbert* and its spin-offs. It is almost a sure bet that in any engineering organization of size in the U.S., a casual observer will find many instances of Adams' cartoons displayed on bulletin boards and in cubicles, perhaps mainly because they illustrate culturally familiar absurdities of intra-firm reality so well. Ever the post-modern, Adams seems especially attuned to the details of power relations and how the logic of power often trumps the logic of propositions.

TRACKING TEAM CONSENSUS ACROSS THE PRODUCT DEVELOPMENT PROCESS

Our research has been inspired by a conjecture of an experienced participant-observer of product development, Don Clausing. Professor Clausing, for a time a chief engineer at Xerox, was among the first to introduce Quality Function Deployment (QFD) to US firms, having studied product development processes at certain world-class Japanese firms. QFD is a matrix-based method for product development teams to carefully consider,

translate, prioritize, and manage the deployment of customer preferences to guide product development from the very earliest marketing activity, through engineering, and into production. (Akao 1990) Done well, QFD reliably ensures that the preferences of customers are well translated into the thought worlds of engineers, designers, manufacturers, suppliers, and managers, and it is for this quite non-trivial purpose that many advocate the use of QFD, Clausing among them. The **Clausing Conjecture** holds that an even greater benefit to expert use of QFD accrues in the deep consensus that its use engenders within the cohort having product realization responsibility, such consensus leading to aligned, coordinated enactment by product realizers. (Clausing 1994) To begin to test this conjecture we must find ways to gauge the state of consensus/dissensus among members of product development teams as the development process unfolds.

Cultural Consensus Theory has been proposed, tested, and elaborated by Romney, Batchelder, Weller, and others. (Romney *et al.* 1986, Batchelder and Romney 1988, others) As defined in the *Proceedings of the National Academy of Sciences*:

> "Cultural consensus analysis consists of a family of formally derived mathematical models that simultaneously provide an estimate of the cultural competence or knowledge of each informant and an estimate of the correct answer to each question asked." (Romney, *et al.* 1996)

In Cultural Consensus Theory, cultural actors are presumed to share knowledge about salient aspects of their culture. Further, some actors are expected to have greater competence about those cultural aspects and can be identified principally because their perspectives lie at or near the center of consensus within the population. This is highly useful for ethnographic field research because it allows for finding of so-called omniscient observers (Boster 1985) within cultures of interest. In particular, if we probe these omniscient observers we can learn more, and more rapidly - it is efficient ethnoscience.

Cultural Consensus Theory has been applied to a number of research questions, for example, folk medical beliefs, judgment of personality traits in a college sorority, semiotic characterizations of alphabetic systems, occupational prestige, causes of death, illness

beliefs of deaf senior citizens, hot-cold concepts of illness, child abuse, national consciousness in Japan, measuring interobserver reliability, to delineate how various cultures construe kinship, and balance risk with desire in sexual practice, among others. (Romney, *et al.* 1996, others)

The concept can be straightforwardly explained through example. Let us suppose that we have given fifty people a multiple-choice examination and then discover that the answer key has been lost. By great coincidence, additional misfortune leads to the burning of all of the question sheets. All we have are the filled-in answer sheets. Cultural Consensus Theory asserts that we may fairly assign grades by discovering the smallish number of people who agree most on the answers, gauging their expertise in terms of inter-respondent agreement (more agreement implying more expertise) and then using their expertise-weighted answers to generate an answer key.

More formally, Cultural Consensus Theory has a basis in three assumptions: (Romney, *et al.* 1986), (Miller *et al.* 1997)

Assumption 1: *Common Truth*

> There is a fixed answer key applicable to all respondents. This is a way of saying that all respondents are members of one culture.

Assumption 2: *Local Independence*

> The respondent-item response random variables satisfy conditional independence (conditional on the correct answer key). In other words, that the answers a respondent gives to a question are not shaped by other questions, or by answers given by another respondent.

Assumption 3: *Homogeneity of Items*

> In the strong version of this assumption, each respondent is assumed to have a fixed competence across all questions. It is also presumed that all questions are of the same degree of difficulty. In a weaker version, it is assumed that the respondents who "do better" with one subset of questions will do better on other subsets of questions.

While proving that these assumptions are valid is not scientifically possible, showing the converse is entirely empirical: Cultural Consensus Theory is specified in such a way that the analyst can determine when at least one of these assumptions is violated. Importantly for us, that assumption violation can be an interesting finding in itself because it suggests that there may be no cultural consensus, or that respondents have connived to manipulate, or that cliques with tight but only local awareness may exist, or some combination.

This empirical finding rests on determining whether or not a singular value decomposition (SVD) of the matrix of correlations between respondents shows proper evidence of having a single cause of variation: the competence of the respondents. This requires that the resulting vector of respondents' competences be entirely positive and that the ratio between the first and second eigenvalues be greater than three. It is beyond the scope of this dissertation to describe various methods of singular value decomposition and their notation and interpretation. As a practical matter, analysts use a number of commercially available software packages to accomplish this. We have used Anthropac (Borgatti 1992) because it provides complete consensus analysis in a compact fashion.

Our research interest diverges somewhat from the above applications and thus our application of Cultural Consensus Theory diverges. Our interest, following the **Clausing Conjecture**, is principally in how (if) consensus changes over time. That is, providing that a relevant perspective consensus exists, if correlation between respondents rises we would say that perspective consensus has increased. If correlation diminishes we would say that perspective consensus has lessened. Clearly, tracking consensus changes across the product development cycle necessitates multiple administrations of our data elicitation

instruments. To reduce the burden of these administrations to both respondents and researchers, we have created computer-mediated versions of several classic instruments of cognitive anthropology.

Application thus far has been to (1.) a group of teams of undergraduate students that designed, built, and raced a 750cc-engined formula car in a national competition, (2.) a group of teams of undergraduate and undergraduate students and faculty that designed and produced prototype tools for collaborative manufacturing, and (3.) a team of professional engineers working to design and produce a derivative product for an established market. The remainder of this dissertation is given to describing our software, these applications, their results, and our conclusions in detail.

CHAPTER 2: THE RESEARCH METHODS

INTRODUCTION

The pile sort, magnitude scaling, and triads test have been used in cognitive anthropology to characterize the way a group of individuals thinks about some aspect of shared reality. Cognitive anthropologists hold that individuals within some sort of culture share (often mostly pre-conscious) schemas, or mental scaffolding, for understanding the world in which they cohabit. Classic applications of the pile sort and triads test include investigating kinship terms and their perceived relationships, human personality temperaments and their perceived relationships, ethnic diversities and their perceived relationships, diseases and their perceived relationships, and societal risks and the perceived relationships between types and kinds of risk. (D'Andrade 1995)

Our research motivation is to determine whether such tools may be used to characterize the way teams of product developers think about their development process and product - more specifically, to understand if, and how, consensus among developers changes over the cycle of product development. In this instance we are primarily concerned with a relatively narrow aspect of consensus: how the team thinks about the development project in terms of importance and difficulty of project activities and, in one case, achieving certain product attributes, too. Thus, we apply the tools of cognitive anthropology not to understand customers as, for example, Stefflre in Chapter 1, but to understand product developers. The remainder of this chapter is devoted to descriptions of pile sorts, magnitude scaling, and triads tests and to our methods of administration and analysis of them.

ELICITATION METHODS

INTRODUCTION

For our research purposes, a three-step method was used. After identifying product development teams possibly willing to participate, informed consent, based principally on anonymity, was obtained from participants. Next, using a variety of methods, the domains of interest were characterized by statements of between three and eight words, each domain having between 18 and 48 statements. Respondents were then presented with various ranking, sorting, and scaling tasks in a variety of media. These presentations were typically performed at the outset, in the middle, and at the end of product development.

SELECTING DOMAIN ELEMENTS

Following informed consent of respondents, our research for a given domain began with elicitation of domain element statements from relevant domain experts - for us, PD team members. Each member was asked to provide a freelist of statements characterizing elements of the domain of interest, such as "all the activities and accomplishments" anticipated in the team's pending development project. Respondents were encouraged to make each statement express only one idea, have fewer than fifty characters, and contain at least one noun and one verb. Because, in our experience, most respondents generated between fifteen and sixty statements, all but one of these freelists were elicited as ASCII text via email to simplify editing and winnowing the statements. The freelist elicited otherwise was generated in the construction of an Affinity diagram (see Appendix B for a description) during which respondents as a group freelisted on PostIt notes, clustered the notes collectively, and collectively titled each cluster; in this case the cluster titles were taken as freelist elements.

A variety of established methods are recommended to distill these freelists into canonic statements of domain element. None is definitive and all ultimately are contingent on the domain of interest, the researcher's *a priori* knowledge of the domain, and questions of

practicality. Much of the effort is best spent on avoiding blunders, apparently, and not on achieving perfection. (Bernard 1994: 237-242; Borgatti 1992: 1-3) For example, where the domain is well-known and well-structured, repetition of expression leads to selection. In our experience, freelisted statements had relatively little literal repetition, but material conceptual repetition, allowing for straightforward paraphrasing. Lists also contained statements at varying levels of abstraction (part:whole) e.g., "design the transmission" vs. "design the car." At our convenience, levels of abstraction were adjusted toward the particular, but only so far that a maximum of fifty or so statements well spanned the domain at about the same level of abstraction. Finally, all expressions were edited to remove the passive voice, to have the same verb-noun-adjective structure, and to include, as much as possible, the verbatim expressions of the team members.

When concern arises as to whether a statement is phrased correctly or makes sense in the culture of interest, the only recourse is to ask culture members. In the case of the team of professional engineers, we chose to do this because the domain had a large amount of culture-specific terminology and perspective. Accordingly, the emailed freelists were written to PostIts. With the assistance of team members' colleagues not currently on the product development team, the meaning of each statement was clarified, essentially duplicate expressions were identified, and all were clustered. We then used the cluster titles and content to produce the list of domain elements, as above. Two of the engineers' colleagues then reviewed the list of domain elements for correct terminology and clarity of expression.

THE PILE SORT

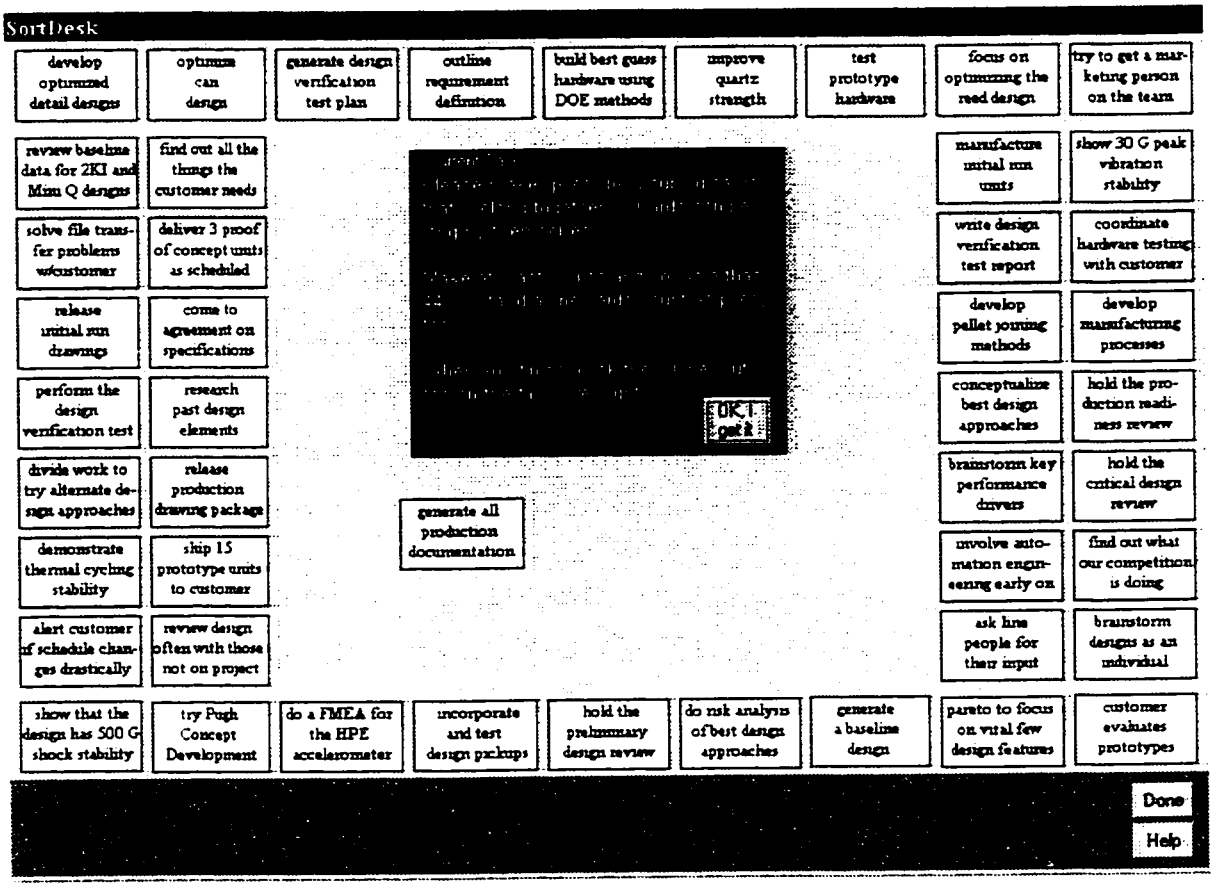


Figure 4: Initial Screen, with Instructions Displayed

A classic way to administer a pile sort (Boster 1994), using cards with one statement printed on each, works as follows: the respondent is given the deck of cards and instructed to make as many piles of "related" or "similar" statements as she wishes, with no guidance given regarding decision making beyond excluding the placing of all cards in one pile or each card in its own. After completing this so-called "free sort," the respondent may be invited to sort further by splitting and combining piles.

We have used Microsoft Visual Basic to produce a PC-mediated version of this sorting process. Our software is called "SortDesk," a title chosen to suggest the convenience of manipulating cards on a large, open desk. In SortDesk, the first task screen presented

shows the domain element statements printed in boxes (visually: "cards") that have been slotted into the startup format in an order that is randomized at every startup (see Figures 4 and 5).

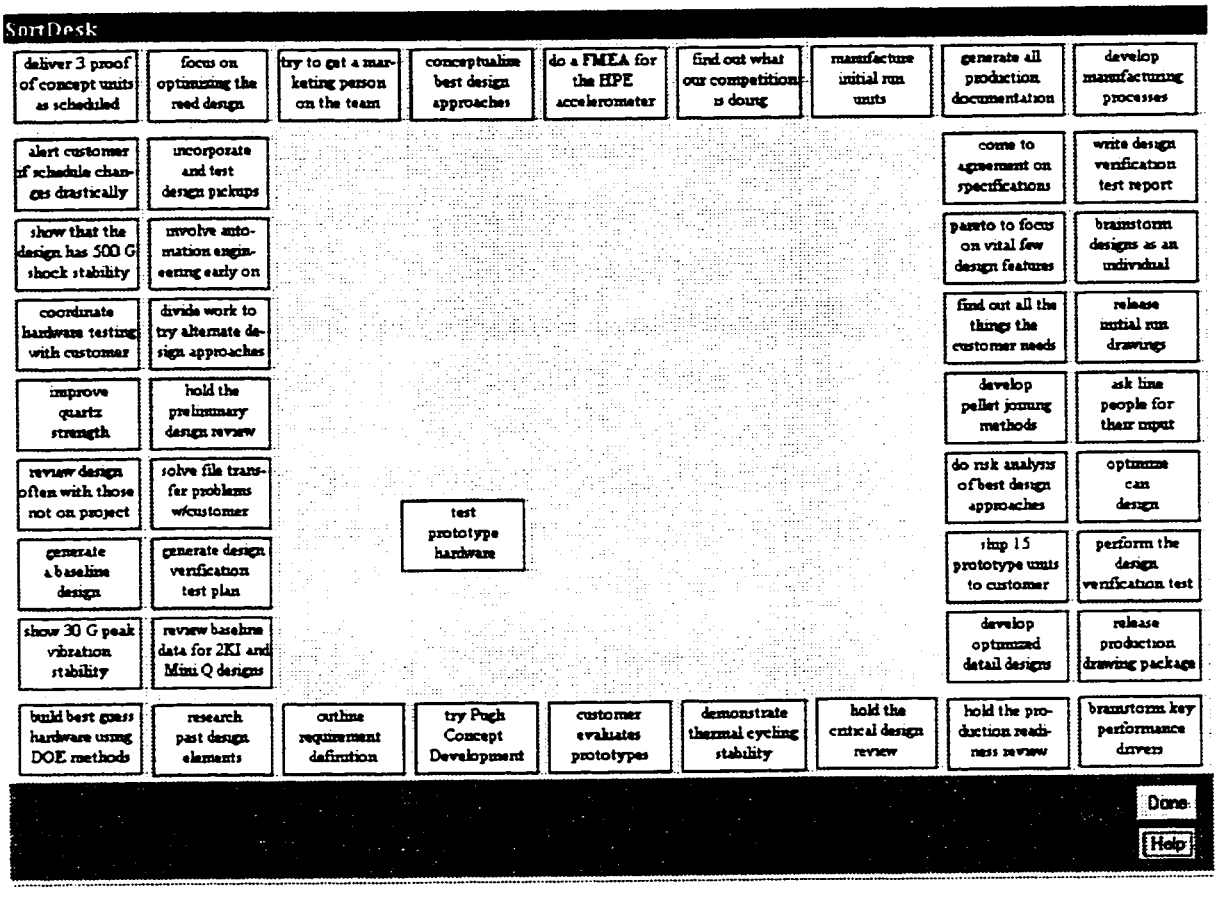


Figure 5: Initial Screen, Instructions Hidden

The respondent next produces piles *ad lib.* by a series of "click-drag-drop" mouse moves. When a single card is dragged, a single card icon moves with the mouse and the card dragged becomes a lowlighted (the visual opposite of highlighted) shell. In the screen shown in Figure 6, the card in the middle is being dragged and its pre-drag location has become lowlighted.

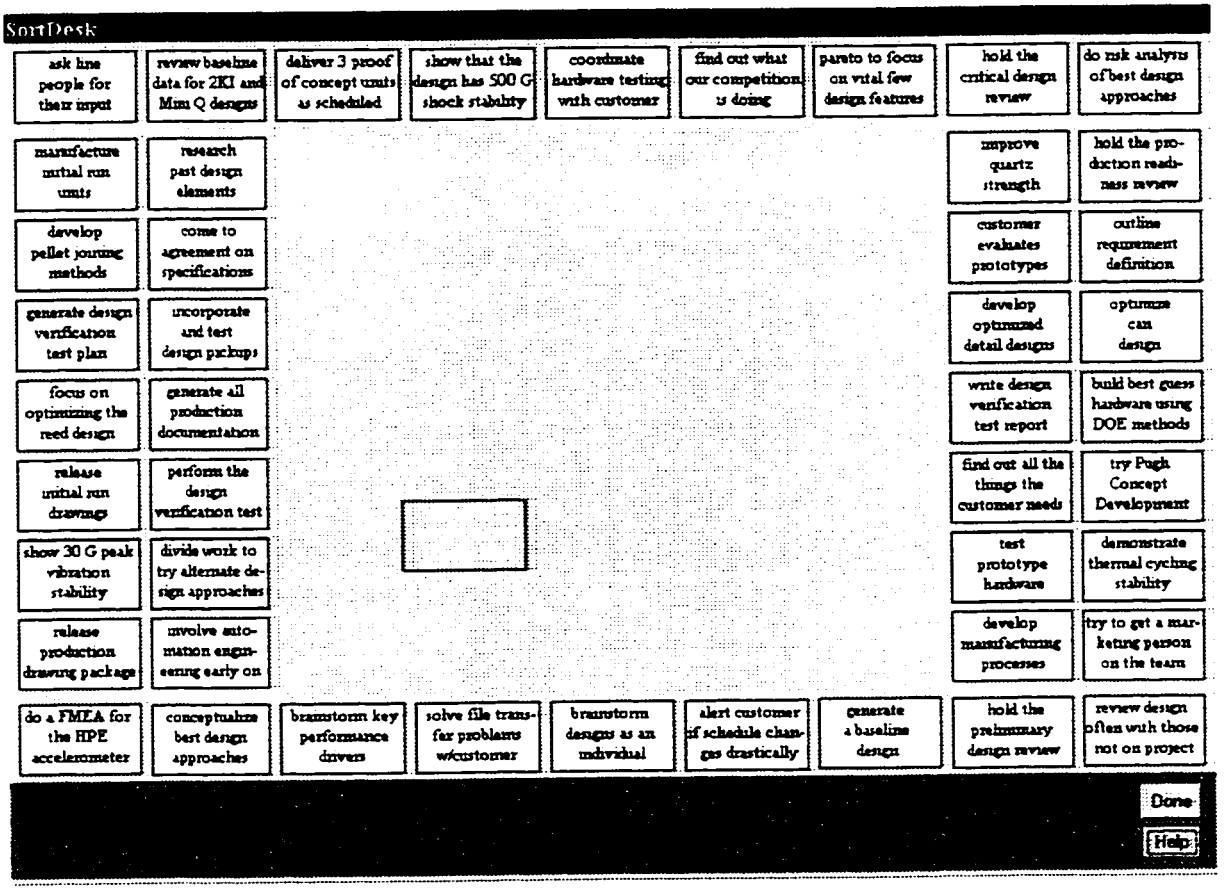


Figure 6: Card Gets Dragged; Old Location is Lowlighted

When a card is dragged over another, the underlying card becomes highlighted in green. In Figure 7 a card from the right middle has been dragged over (and remains "flying" over) the card in the center, causing the card in the center to be highlighted in green. If a flying card is dropped (the mouse button released) over a green highlighted card, the flying card "snap-joins" the highlighted card and the two share the same pile. The snap-join occurs at the edge of the highlighted card that is nearest to the flying card's drag icon, in this instance the right edge. (See Figures 7 and 8) Cards not snap-joined do not share the same pile no matter how close - even overlapping each other - they lie on the screen.

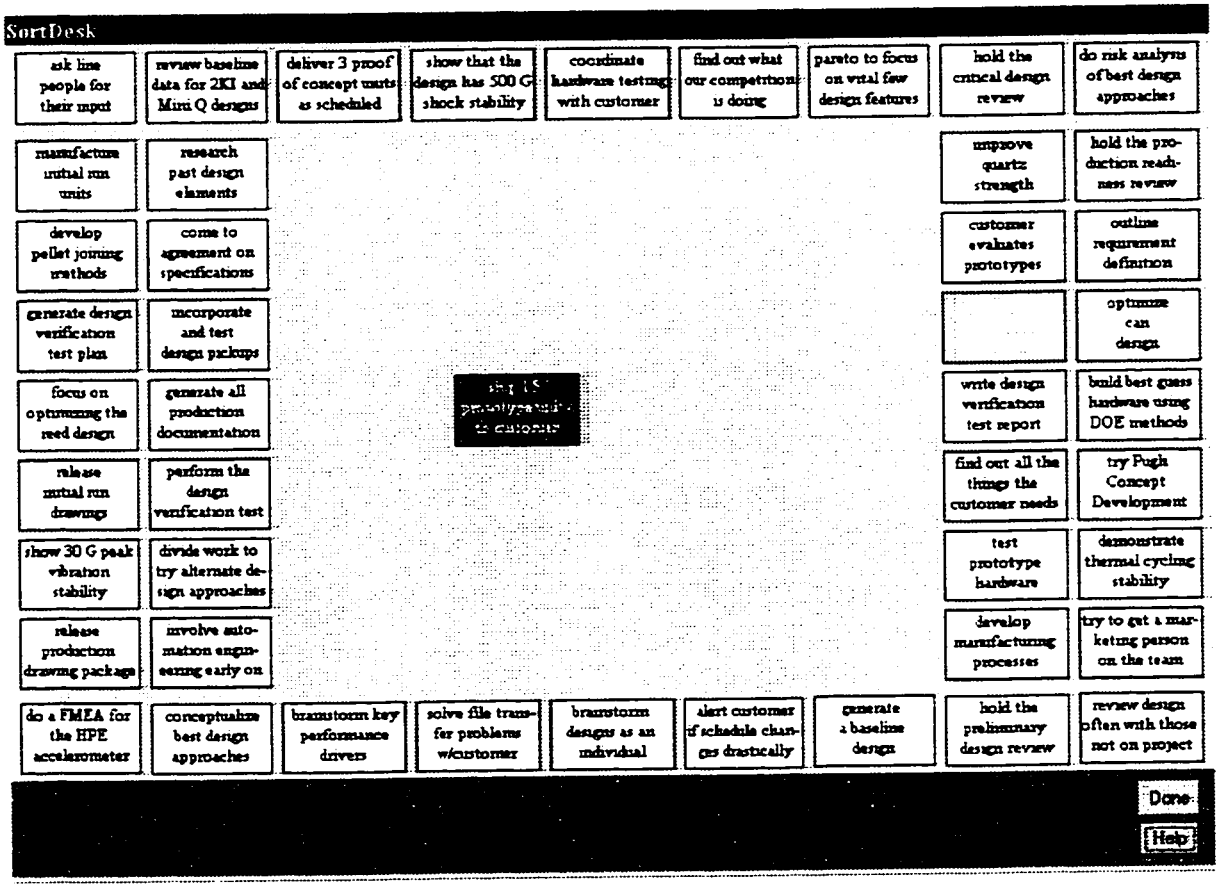


Figure 7: Dragged Card Highlights Underlying Card in Green

Cards can be removed from piles by simply dragging them from the pile. Entire piles can be moved by first "double-clicking" on any card in the pile, which highlights the entire pile in green (this method may also be used to determine precisely which cards are in a given pile). When a pile is highlighted in green, dragging any card in the pile drags the entire pile. (See Figures 11 and 12.)

When the dragged pile's drag icon "flies" over another pile, the underlying pile highlights green, as in Figure 13.

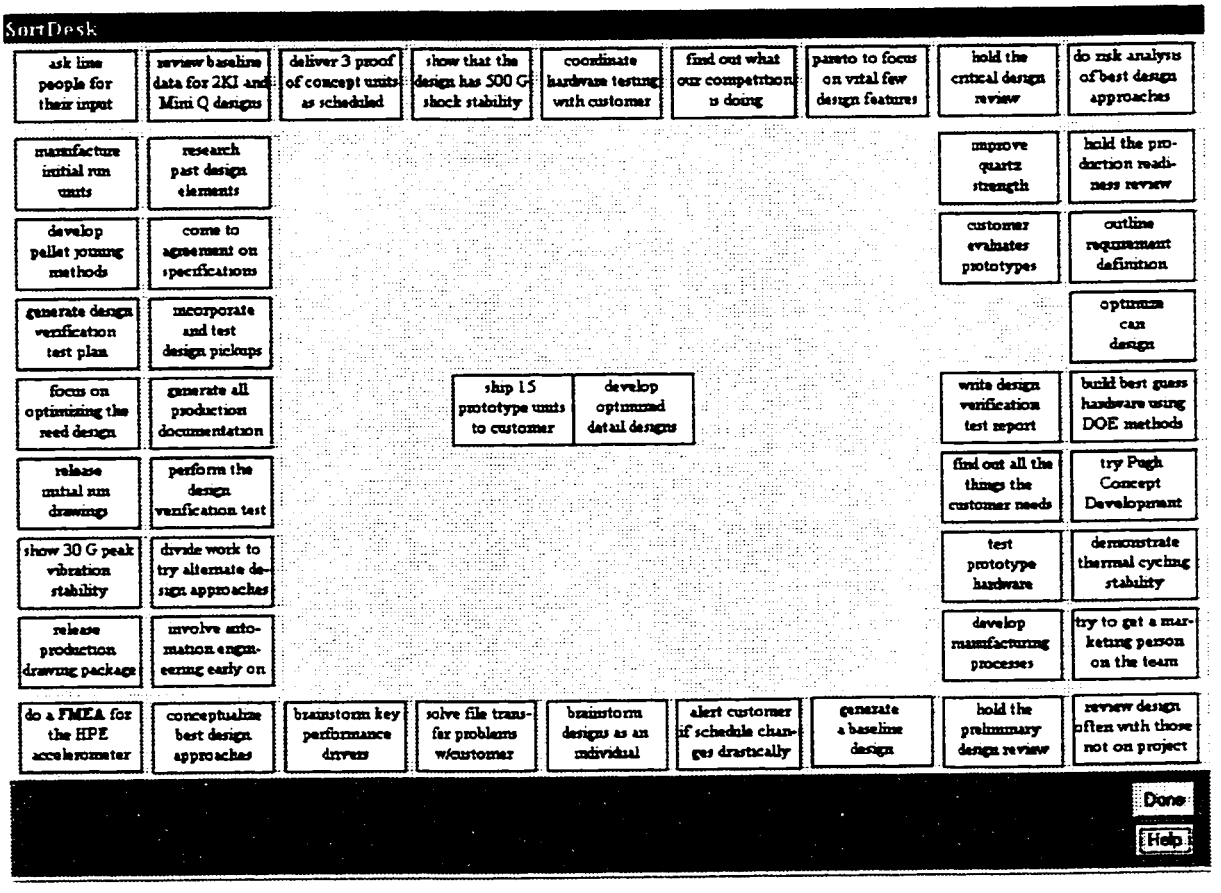


Figure 8: Cards Have Snap-Joined

If the dragged pile is dropped over the underlying pile, the dragged pile's cards are tiled in a spiral fashion around the underlying pile. The spiral begins at the card edge nearest the drag icon when the drop occurs and winds around the underlying pile in a clockwise fashion. (See Figure 14.)

At the end of the free sort, the respondent will have distributed snap-joined piles around the screen. Upon completion of the free sort she clicks the Done button on the lower right. (See Figure 15.)

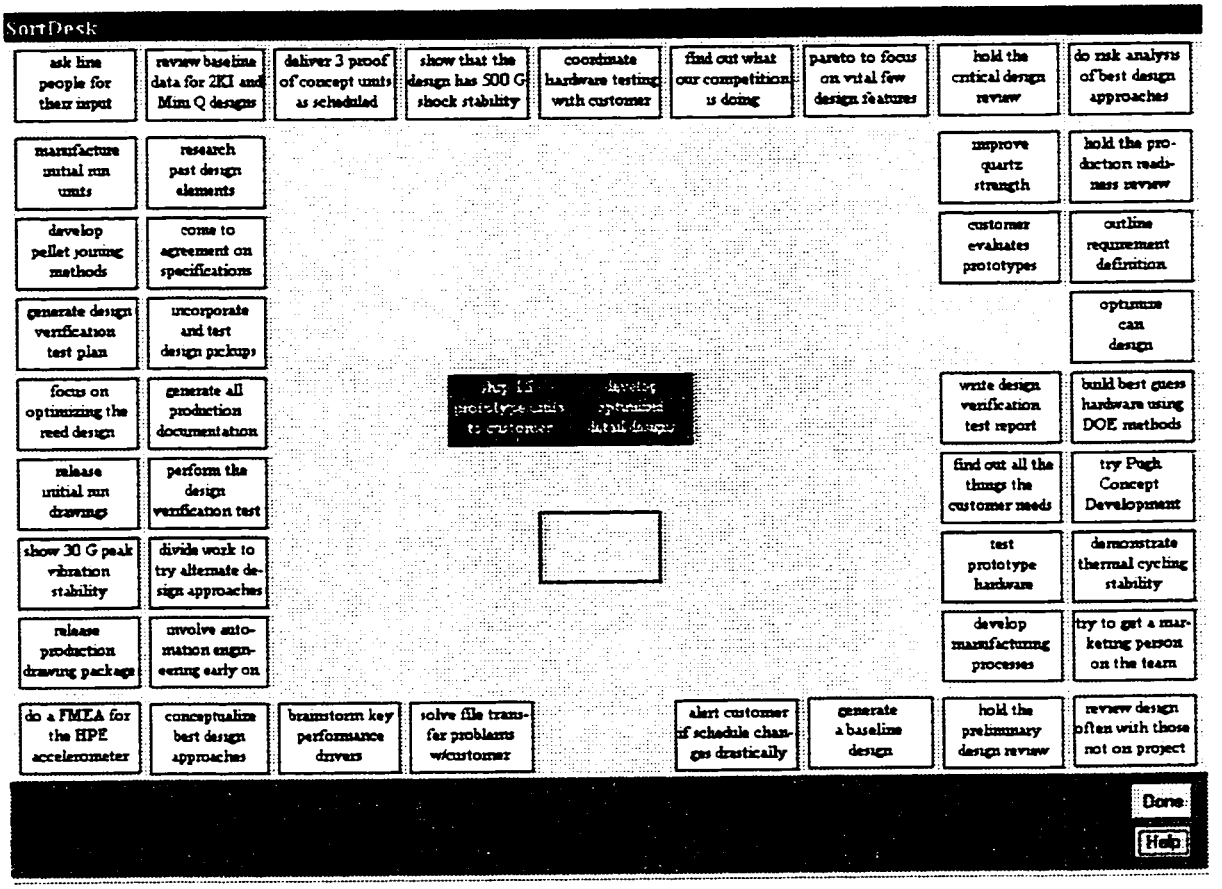


Figure 9: Card From Screen Bottom Dragged
Over the 2-Card Pile in the Center

After clicking the Done button, the respondent is prompted, pile by pile, to offer a rationale for ascribing similarity to statements in the pile (See Figure 16.). If she wishes, she may enter a rationale in the text box by keying it in, she may click on one or more of the cards in the pile and have the card(s) statements(s) automatically entered in the text box, or she may simply skip the entry of this pile's rationale. After providing a rationale or not, she clicks the Enter Text button to the right of the text box and the next pile is highlighted to prompt for its rationale's entry.

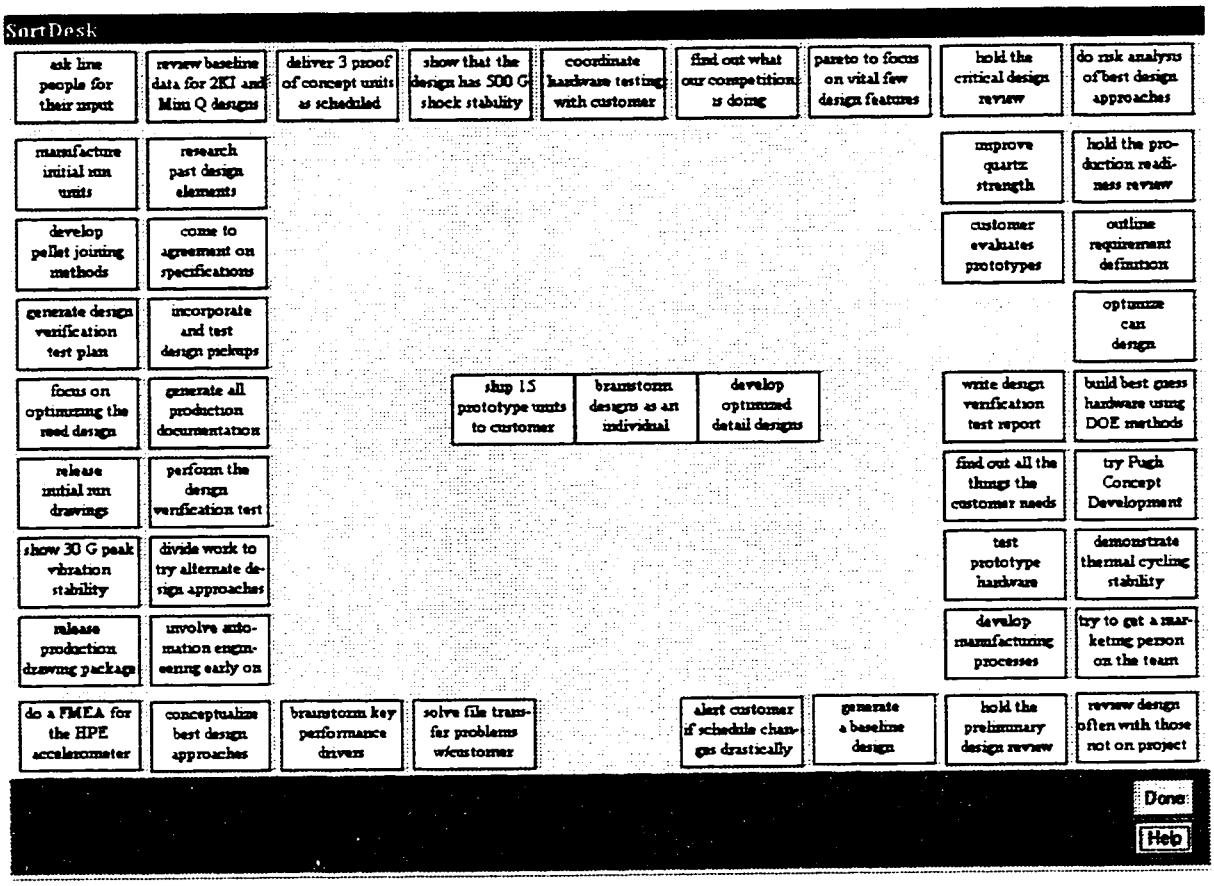


Figure 10: Card From Screen Bottom Has
Bumped Right Card in Pile Rightward

In the second, optional segment of the sort the respondent is asked to create an additional pile by dividing one of the piles she created in the free sort into two piles, each with some similarity of its own. When she finishes splitting and entering rationales for these piles, the changes in pile membership and rationales for the two new piles are recorded. The respondent continues to split piles in a stepwise fashion, and the software to record pile membership and rationales, until all cards are in singleton piles or until the respondent wishes to conclude this segment of the administration.

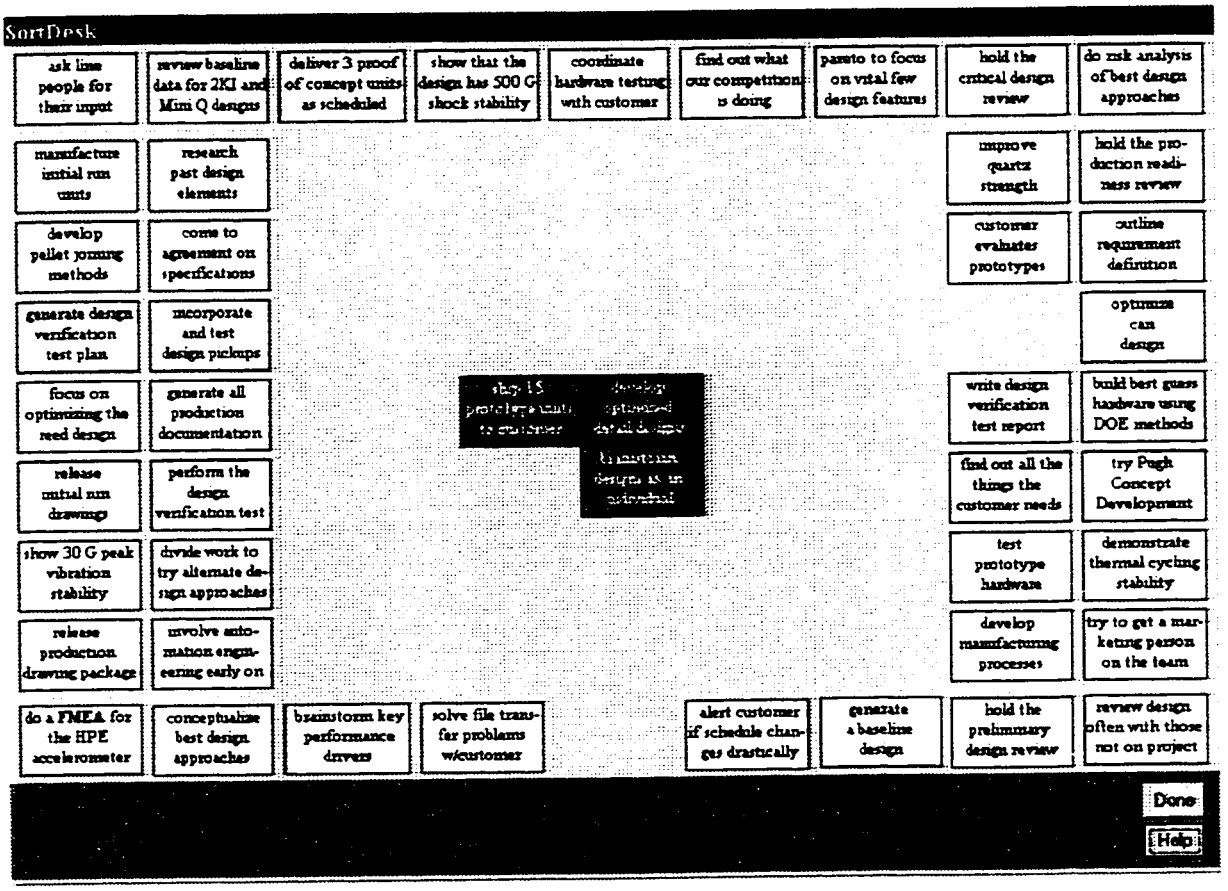


Figure 11: A Green Highlighted Pile in the Center

In the third (and optional) segment of the sort, the cards are returned to the state of the initial free sort. The respondent is asked to create one fewer pile by combining two piles from the initial sort and to offer a rationale for the new pile's similarity. The respondent continues to combine piles, and the software to record pile membership and rationale, until the cards are in two piles or until the respondent wishes to conclude this segment of the administration.

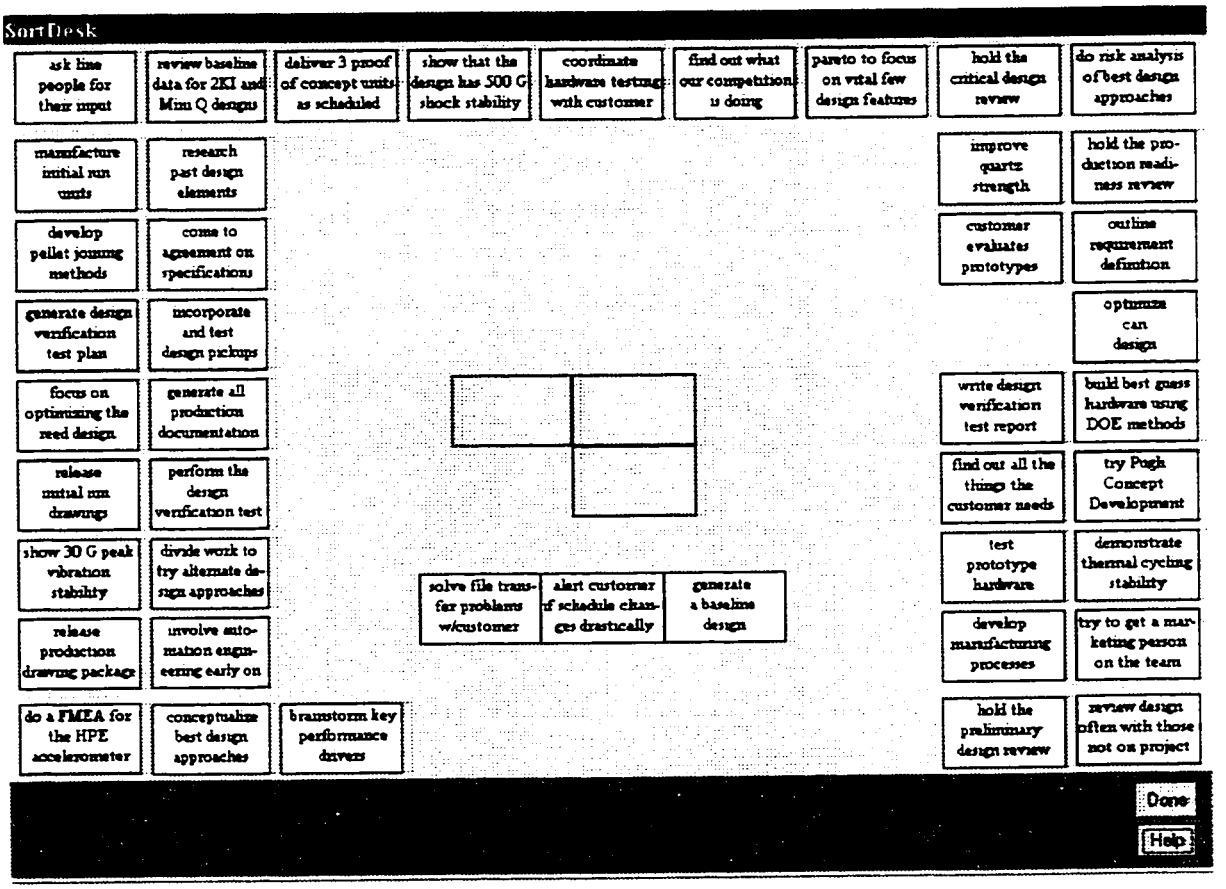


Figure 12: Entire Pile From Center is Dragged

MAGNITUDE SCALING

The next two segments of the administration ask the respondent to distribute the domain statements along two preselected dimensions: Importance and Difficulty. Often such distribution is done by presenting lists of statements and asking respondents to assign a weighting to each statement based on a Likert scale, typically scaled from 1 to 7 or 1 to10. One of the drawbacks to this approach comes from restricting responses to categorical values and another is the difficulty of maintaining consistency and transitivity across a large number of domain items - it's simply too hard for respondents to scan all of what they've distributed to see how they've distributed bits of it. We take advantage of PC-mediation to present each distribution task as one of placing the statement cards,

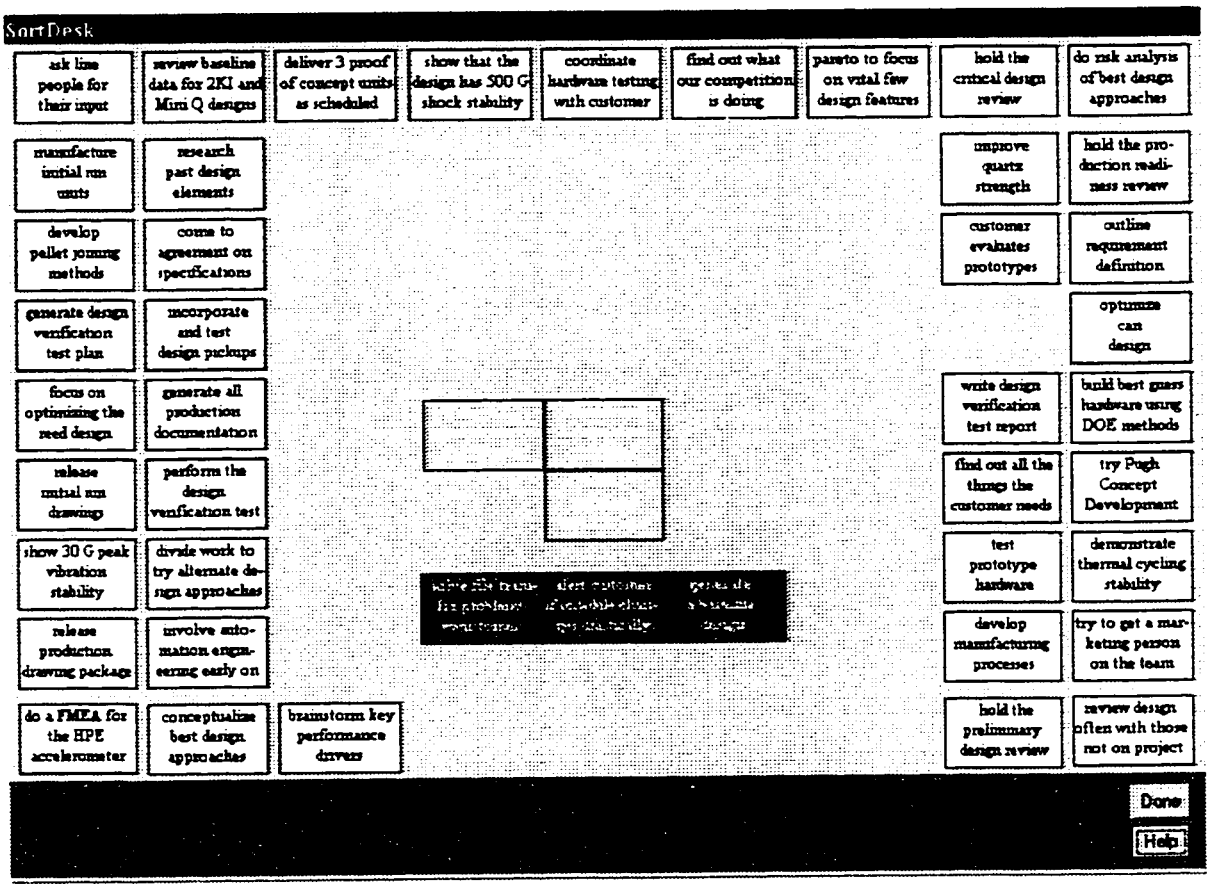


Figure 13: Dragged Pile Highlights Another Pile

from left to right on the screen along the dimension scaled. Two benefits flow from this. First respondents can more easily keep all the statements in view, thus aiding consistency and enforcing transitivity. Second, because the screen is divided by Visual Basic into what is effectively a continuum of roughly 12,000 units from left to right, we may more safely do analysis based on assumptions of magnitude (interval-scaling) rather than ordinality.

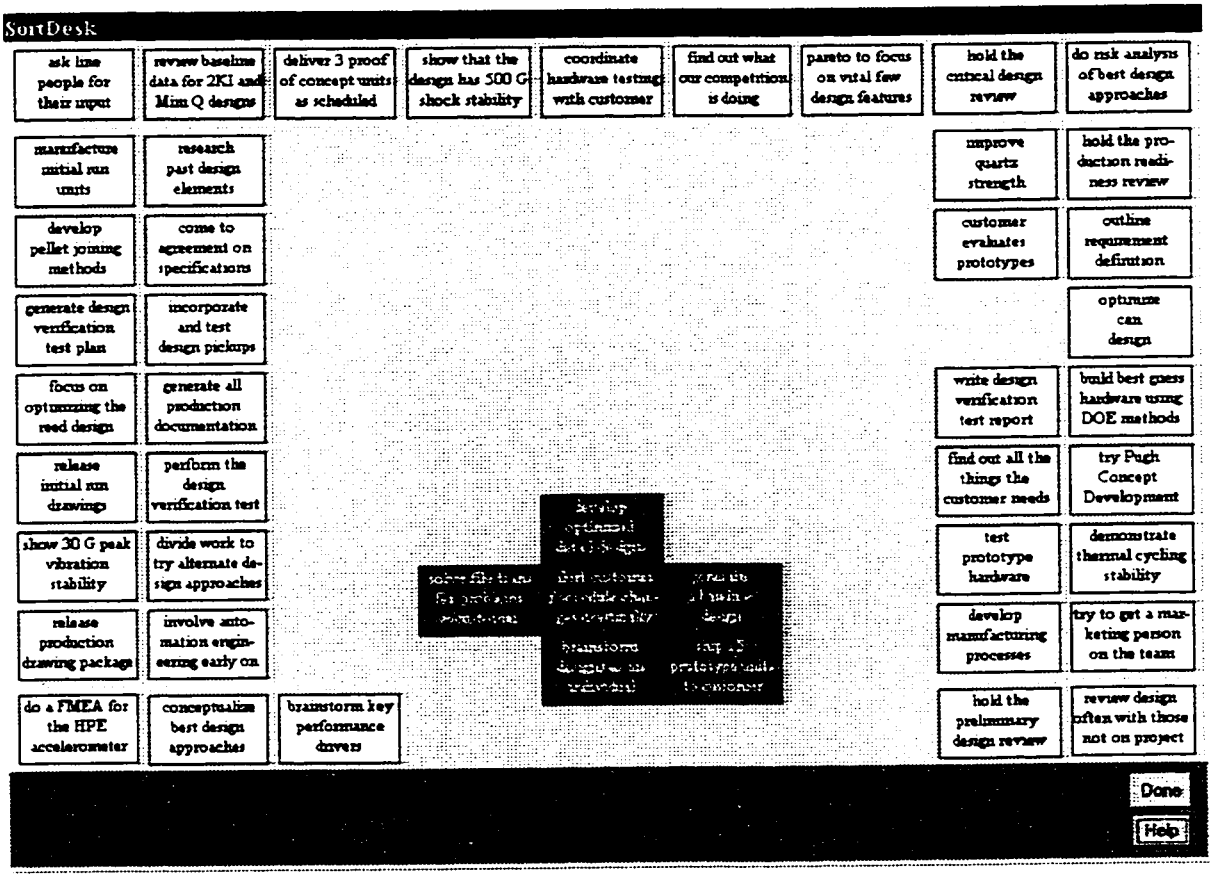


Figure 14: Dragged Pile Snap-Joins Underlying Pile by Spiral Tiling

In this segment of the administration, the cards are placed in a single deck at the bottom center of the screen, and the dimension along which respondents will scale is displayed across the top of the screen. As in the pile sort, the order of the cards in the deck is randomized during startup of the display for that segment.

In Figure 17, the initial Importance scaling screen shows these features and the instructions for this task. Respondents drag each card from the deck and position it on the screen according to its importance. Adjustments for changes in relative importance are easily made as new cards appear on the deck top and it is easy for respondents to detect emergent inconsistencies. If a respondent wishes to assign exactly equal importance to two or more cards, she may take advantage of the snap-join feature to stack cards vertically as stacked cards have the same left-to-right screen coordinate and thus are

scaled as equally important. Figure 18 shows what the screen might look like upon completion of this task. For the sake of brevity we will not discuss or show Difficulty scaling because it is entirely analogous to Importance scaling.

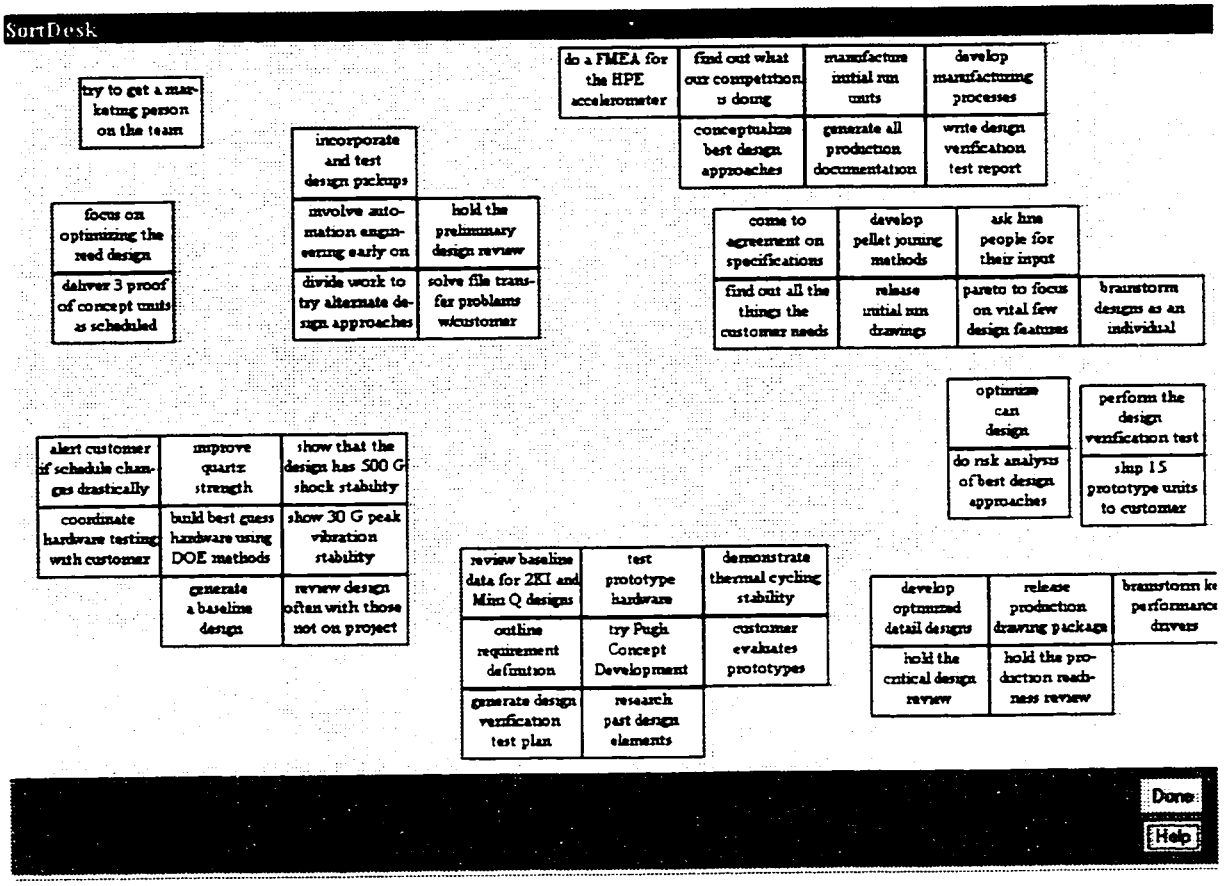


Figure 15: Screen After a Typical Free Sort

TRIADS TESTS

Because we wish to characterize dispersion in pile sort data as diversity of perspective, we would like to be able separate "within respondent variance due to cognitive imprecision" (an error variance) from "between respondent variance due to perspective differences," and both of those from "within respondent variance due to change of perspective." One way of doing this uses administration of two different appearing but functionally

equivalent techniques in relatively rapid succession. In our application, we attempted to pair a triads test with a pile sort; each allows estimation of Euclidean distances between elements of schema. When a triads test is administered, the respondent is presented with a series of schema statement triplets and asked to indicate which of each triplet most differs from the other two. Using our SortDesk software, we also mediate the triads test via PC, presenting three cards at a time on the screen. Respondents are asked to click on the card that belongs least.

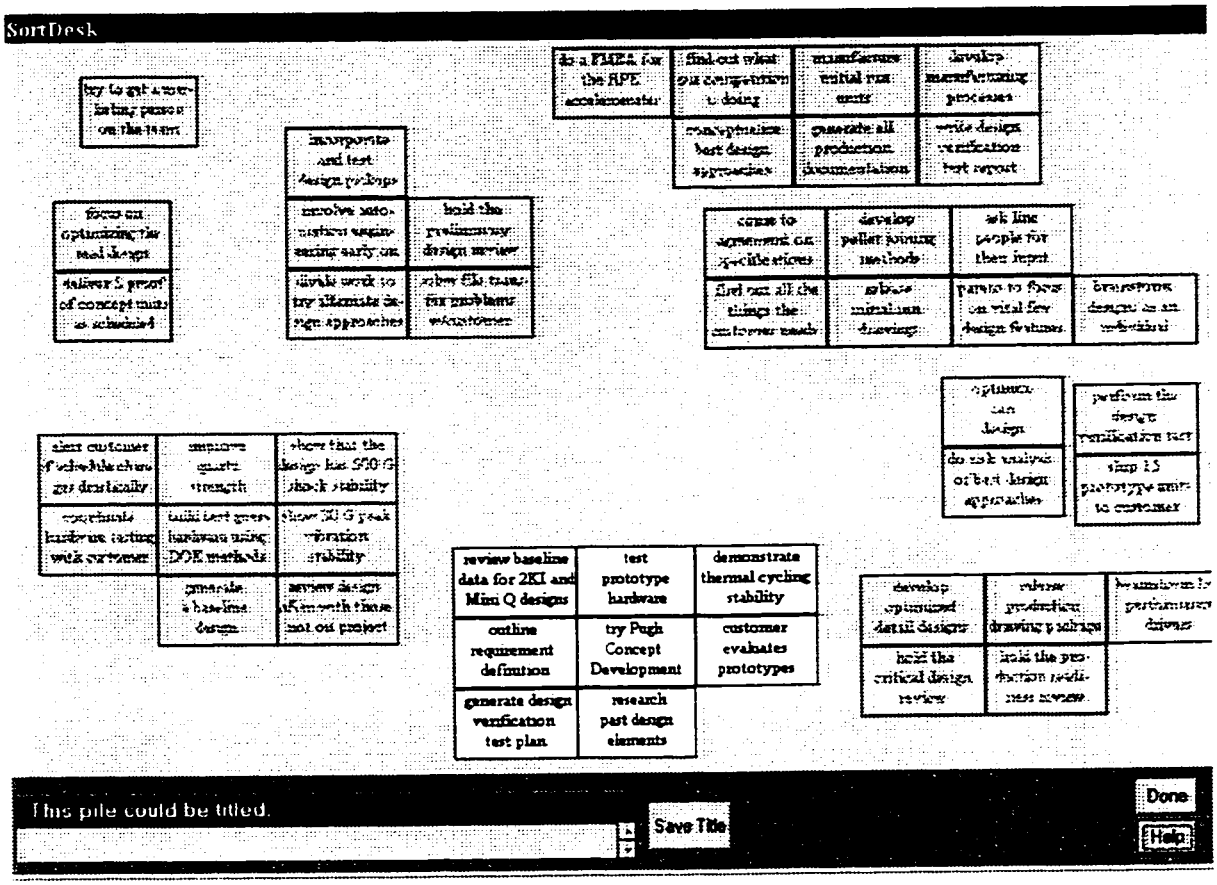


Figure 16: Prompting for Rationale of Highlighted Pile

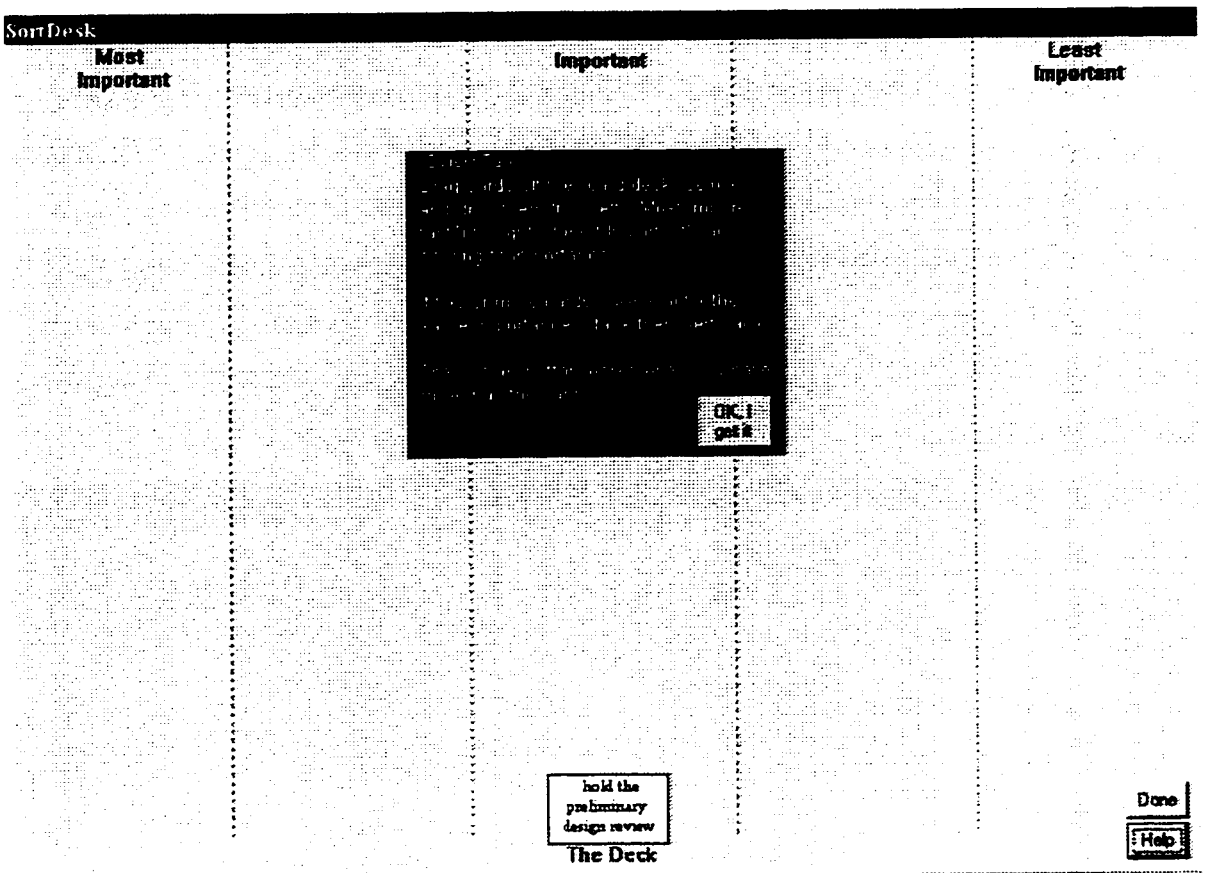


Figure 17: Initial Importance Scaling Screen, with Instructions

Triads tests may be administered as full factorial investigations, with every pair of statements being presented with every remaining statement, but such requires a combinatoric explosion (O(n!)) of respondent discrimination. Consequently, triads tests are often administered as balanced incomplete block (BIB) experimental designs. BIB designs reduce the number of discriminations required of any respondent but also reduce the reliability of results. Researchers can compensate for this by administering to a larger sample of respondents.To minimize respondent burden, and having domains with larger numbers of element statements, we elected to administer a lambda 1 balanced incomplete block experiment, the most highly fractionated and least reliable design. In a lambda 1 design, each possible pair is presented only once, contrasted with a third statement that is not randomly selected. In the one instance that we administered a triads.

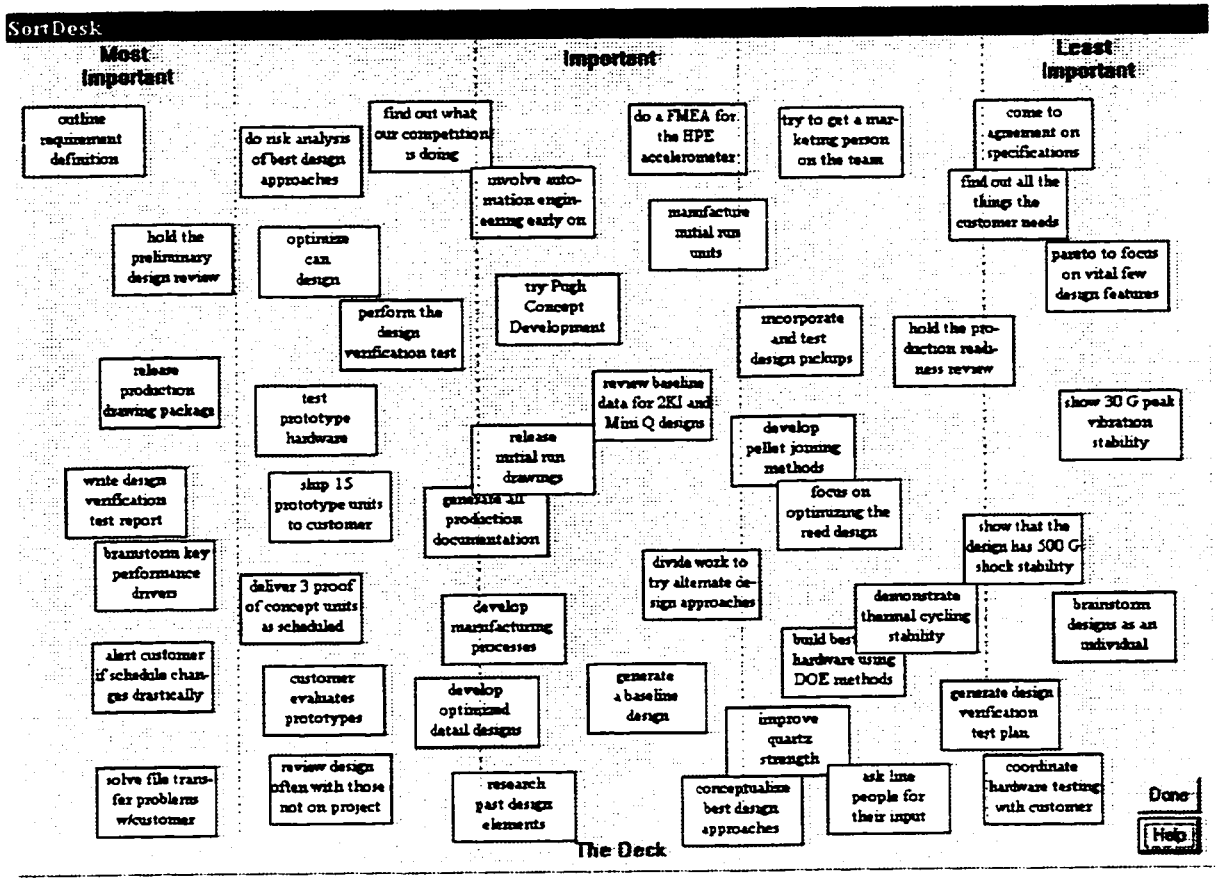


Figure 18: Screen on Completion of Typical Importance Scaling

test, the domain investigated had only about half the number of elements (25) as our average domain. Even after halving the potential dimensionality of the experiment, respondents would need to make 2,300 discriminations in a full factorial triads test. Using a lambda 1 BIB design reduced the requirement dramatically to 200 triads. It is well to note here that triads tests may seem minimally burdensome, perhaps because discriminating between only three domain elements seems simpler than between many at once. Further, BIB designs can reduce repetition burden by an order of magnitude.

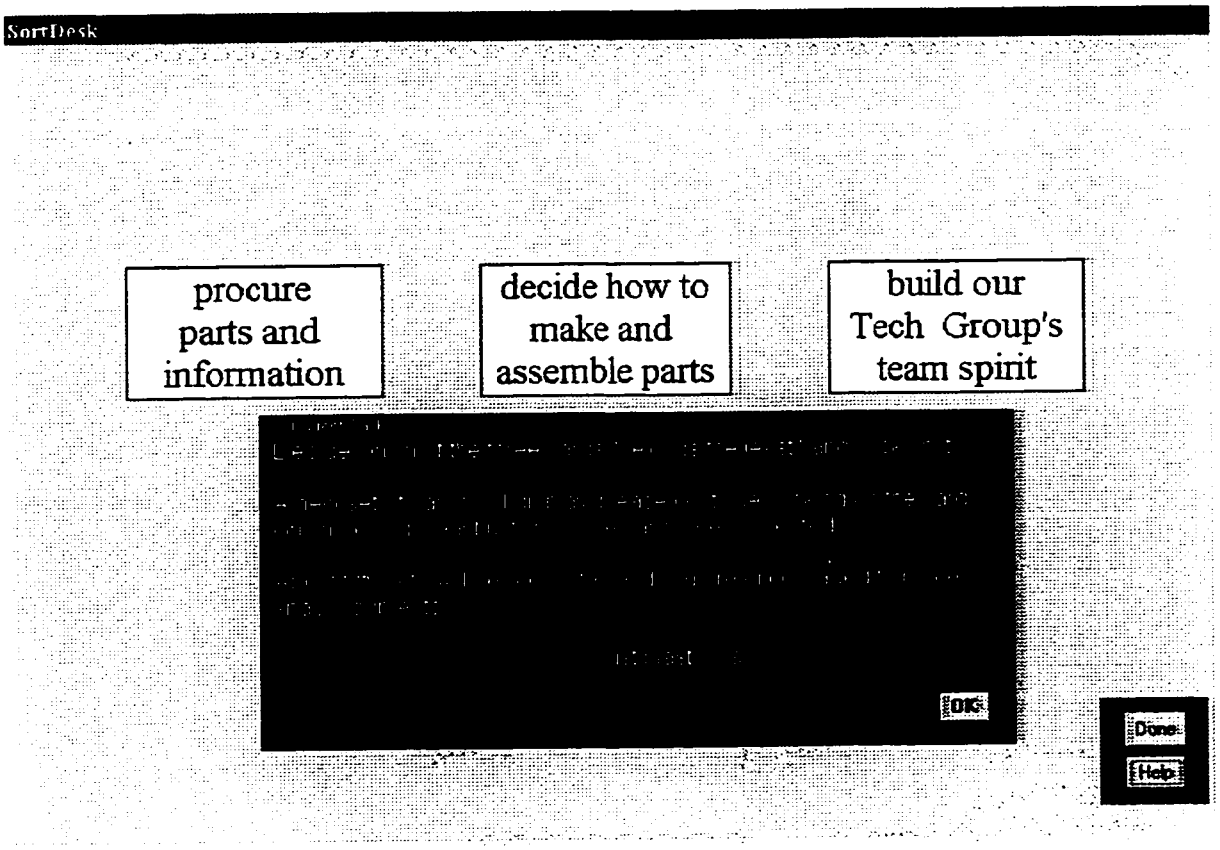


Figure 19: A Typical Triad Presentation, with Help Window Displayed

PC mediation of triads tests is straightforward: a program displays, in random order, triads called out by the BIB design. Because only three statements need be displayed simultaneously, their card embodiments can be large and read easily at some distance. In our PC version (see Figure 19) the card triad is centered vertically and horizontally on the screen. To further minimize respondent burden, no rationale is sought for each discrimination. Clicking on the least similar card causes each of the triad's card numbers (and another copy of the chosen card's number) to be written to a line in the data file. After the write, the next triad is presented.

CHAPTER 3: THE INSTRUMENT ADMINISTRATIONS

INTRODUCTION

We have collected data from three distinct groups of product developers: one composed of professional engineers and technicians and two primarily composed of undergraduate engineering students. The first group includes four degreed engineers, one technician, and one marketing specialist. Of the two groups of academic origin, one consists of 42 engineering undergraduates and the other has 27 engineering undergraduates, 4 graduate engineering students, 4 engineering professors, and 2 high school science educators. By definition, all included respondents completed at least one discrimination task set at least once.

While each group had a product development task, only the team of professional engineers clearly represents a (small) sample of commercial product development. The other two teams' selections had much to do with availability and accessibility; it is easy now to appreciate why so much research utilizes students as respondents. Nevertheless, many of the questions this research investigated do not apply solely to commercial development and can be addressed with the populations that were instrumented.

This chapter is divided into four sections. Following this introduction is a description of the administrations of the various instruments to the PD teams. Section 2 details administration to the commercial product development, ASC Derivative Product Development, Section 3 details the instrumentation of SAE Formula Car '97, and Section 4 describes the largest of the three instrumentations, the Summer 1997 Special Project teams. Table 1, below displays the (up to 6) administrations schematically.

In Table 1 the following abbreviations are used:

- card IMP: 8 element ordering based on importance, card deck
- sd BSRT: 4 element full Boster w/rationale, Sort Desk
- sd SRT1; 29-38 element free sort w/rationale, Sort Desk
- sd TRI: 24 element lambda 1 triads test, Sort Desk
- sd SRT2: 24 element free sort w/rationale, Sort Desk
- p&p IMP: 18 element ordering based on importance, paper & pen
- sd SRT: 39-46 element free sort w/rationale, Sort Desk
- sd IMP: 39-46 element importance scaling, Sort Desk
- sd DIF: 39-46 element difficulty scaling, Sort Desk

Table 1: Methods, Media, and Team by Administration

		SAE '97	Summer '97 Special Project	ASC Derivative
ADMINISTRATION	1	card IMP	sd SRT, sd IMP, sd DIF	sd SRT, sd IMP, sd DIF
	2	sd BSRT	sd SRT, sd IMP, sd DIF	sd SRT, sd IMP, sd DIF
	3	sd SRT	sd SRT, sd IMP, sd DIF	
	4	sd TRI		
	5	sd SRT		
	6	p&p IMP		

SAE FORMULA CAR '97

Between the fall of 1996 and summer 1997, 42 University of Washington students (working for college credit) organized to fund, design, manufacture, assemble, test, and race a 750cc single seat car in the Society of Automotive Engineers Formula Car '97 competition, an important annual national competition. In May 1997 more than 100 teams

of undergraduates competed in Detroit, under the watchful eyes of automobile manufacturers. It may be important to note that in the winter of 1996, when Ford Motor Company recruiters visited the University of Washington, they elected to interview only students who had participated in a Formula Car competition, providing special incentive for student participation. (Calkins 1996)

Though hardly an experienced automobile design/build/race team, these students confronted problems perhaps similar to such a team and had possibly greater zeal than an average undergraduate (they certainly put significant amounts of time into the project). According to their resumes, many had substantial automotive or racing experience. Each student joined two teams, one among the six technical groups and one of the five administrative groups. Over the course of eight months, two importance rankings, three pile sorts, and one triads test were administered to this group of respondents.

Domain elements were generated for the importance rankings using an Affinity diagram process completed by the SAE Formula Car '96 team shortly after their return from the competition in May of 1996. (See Appendix B for description of the Affinity diagram process.) From the diagram, 18 statements were constructed that balanced consistency of expression, preference for the verbatim, and compactness.

Each of the first importance ranking administrations occurred some time in the first month of the first quarter of the project, immediately after each of the informed consent interviews required to satisfy human subjects research requirements. In this administration, statements were presented on 3"x5" note cards in a freshly shuffled coded deck. Respondents were asked to consider the whole project on which they were embarking, not just the present and near term, and "put the cards in order of importance." No other ranking instructions were given. Rank order of placement was manually recorded in an Excel spreadsheet after completion of each ranking. 31 students completed this task.

We administered the second importance ranking at the end of the third quarter of the project, shortly after the students returned from the competition. At the beginning of class, students attending were given a sheet of paper with the statements in two side by side columns of nine each. They were asked to think of the advice they'd give next year's team and to rank the total project importance of the 18 statements by writing a unique number between one and eighteen next to each - in effect, to order the statements as they had with the card deck. 24 students completed this task, 14 of whom also completed the first importance ranking.

Domain element statements for the first and third pile sorts (and for the triads test) were generated by asking respondents to freelist "All the activities of the {appropriate tech group title} technical group." The lists were solicited and received by electronic mail. By methods outlined above in Chapter 2, each tech group's freelists were condensed into a list of group-specific statements. Presuming that each tech group had technical system design as a primary activity and taking advantage of our expertise in the product development domain, the technical groups' lists were condensed into a single list of 24 generic "technical development" statements that applied to all technical groups. Because the triads test required 25 statements for lambda 1 experimental design balance, an additional statement ("combine analysis with direction") was crafted to sound domain-relevant but be as content-free as possible. Addition of this essentially meaningless statement allowed potential for some assessment of what random response might look like in the analysis.

Students were also organized into administrative groups that had responsibility for all the non-technical aspects of the project. Domain element freelists for the admin group pile sort were elicited as for the technical group pile sort. Admin group freelists were condensed into group-specific statement lists but not condensed into a single generic "administrative development" list. Accordingly, separate versions of the data elicitation software were prepared for each administrative group, each with its own statements.

The protocol of administration for the first technical development pile sort was based on extension of the successive sort advocated by James Boster. (Boster 1994) In this sort respondents are first asked to free sort *ad lib.*, then asked to combine piles stepwise until two remain, then the cards are returned to the free sort state and respondents are asked to split piles stepwise until each pile is a singleton. Our extension added, at each stage in the protocol in which any piles are resorted, solicitation of rationale for each changed pile. In our version text entry is optional via a standard text box and element statements may be click-pasted when highlighted. Disappointingly, only about a third of the respondents completed the "Full Boster" task.

The second pile sort presented "administrative development" in group-specific domains of between 28 and 39 element elaboration. We continued to request rationale, but reduced the task to a single free sort. About a third of the respondents completed the task; of these, a minority had completed the first pile sort.

The third pile sort again presented the "technical development" domain elements as a single free sort with request for rationale. About a third of the respondents completed the task; three of these had completed the first technical development pile sort.

SUMMER 1997 SPECIAL PROJECT

The second academic-originating group of respondents consisted of 32 undergraduate engineering students, four graduate engineering students, four professors, and two high school science teachers from a variety of schools in the United States, who gathered at a U.S. university for 90 days in the summer of 1997. Largely self-organized into five "companies," respondents had to work together to attempt to find ways to relate to each other as enterprises (e.g., exchanging products and services), and to external commercial and governmental enterprises. Beginning with a minimal and oblique hierarchy, respondents focused on developing products and services to support collaborative

manufacturing and had hopes of demonstrating a value addition chain operating in a largely virtual or electronically mediated way within the 90 days.

Per the intent of the organizers of the overall project, as little direct guidance was given to company teams as possible. Respondents frequently expressed frustration with lack of clarity about goals for them as individuals and teams. (Monahan 1997) Given that most respondents had been selected for achievement as undergraduate engineering students - for doing good, detailed work with relatively clear requirements - some dissatisfaction cannot seem surprising,. Other challenges that the company teams faced included lack of expertise with the technology required, lack of expertise in team product development, lack of interpersonal familiarity (commercial PD teams rarely are made of so many relative strangers, and of team members who have such a short time to interact), and resource and time constraints.

Domain elements for each of the five teams were generated from freelists elicited from each team's members via electronic mail. Respondents were asked to list "all the activities of the {appropriate company title} team." By use of methods outlined above, each team's lists were condensed to between 39 and 44 statements specific to that team. Separate versions of SortDesk were built for each team.

Sorting, importance ranking, and difficulty ranking were administered three times: once shortly after team formation, once midway through the project, and once at project end. Respondents were given no special compensation for participation and were not sanctioned for non-participation other than having to bear the researchers' importuning.

Initial response was excellent, with all but four respondents completing the sort and the two scalings for their domain. On second administration, compliance dropped slightly, with all but six respondents completing the full administration. At the final administration compliance dropped more substantially, with all but ten respondents completing.

ASC DERIVATIVE PRODUCT DEVELOPMENT

We instrumented an experienced team of aerospace electromechanical design engineers, marketing staff, manufacturing engineers, and test specialists who designed and produced prototypes of a derivative product. The team was comprised of expert, familiar, and congenial respondents operating in a routinized, rewarding, and moderately pressured environment with much autonomy. Team members had experience with use of Affinity diagrams and the team's leaders had interest in applying consensus analysis.

Domain element statements for the first pile sort and importance and difficulty scalings were generated by first asking respondents to freelist "All the activities and the accomplishments of the HPEA project." The lists were solicited and received by electronic mail. By methods outlined above, the team's freelists were condensed into a list of group-specific statements. Then a convenience sample (n = 2) of reviewers, drawn from engineers from the same organization but not part of the HPEA project, was used to further refine the list to 47 putatively canonic statements. These reviewers helped to ensure proper use of jargon and to harmonize abstraction-level differences, with the intent to leave the domain well spanned and divided with even granularity.

The first sort and scalings were completed within one month after project initiation. The final sort and scalings were completed six months later, after the prototypes were shipped. All but two respondents completed all administrations.

CHAPTER 4: ANALYSIS

INTRODUCTION

This chapter divides analysis into two categories, Respondent Administration Burden and Consensus Discrimination. The second section of this chapter reports on issues of burden, i.e., respondent compliance, average times to complete, and anecdotal evidence. The third section applies a variety of techniques to graphically display or tabulate changes in possible metrics of consensus states; in this section, links are suggested with anecdotal data.

RESPONDENT ADMINISTRATION BURDEN

Inspection of the results of the single administration of the triads test shows that respondents did not like the triads test: only six respondents completed it. It was so negatively received on first administration that it was not inflicted on respondents subsequently. Even though most felt that considering the similarity of only three cards at a time was a much lower cognitive burden than considering forty, the number of presentations required for inferential validity quite soon tried their patience. Apparently, it seems easier (and quicker) to sort "in the large" - with "all your cards on the table" rather than to mechanically make a series of more numerous but more trivial judgments. It is useful to note that respondents found excessively burdensome even the minimum number of presentations (about 200) required to make reasonable inferences from a twenty-five statement domain when using a balanced incomplete block experiment design. (Burton and Nerlove 1976) When domains have upwards of forty statements, the combinatorics of triads tests must finally defeat even the cleverest experimental design. A full factorial

presentation would require (${}_{40}C_3$ =) 9,880 presentations, and as far as we know, no one has attempted to generate a balanced incomplete block design for this many domain elements as it is apparent that even a ten-fold reduction in presentations (a typical amount) would still represent too burdensome a task for most respondents.

It would seem that triads tests have limited utility for application to PD teams. In our SAE Formula Car '97 experiments, very few respondents completed both a pile sort and the triads test, so direct comparisons are problematic. However, for those who completed both, the triads test took longer than pile sorting a similarly sized domain, for some half again as much time. Given the dimensionality useful for investigating product developer domains, pile sorting has significant advantage in administration time and respondent burden.

Table 2: Summer 1997 Special Project Mean
Times to Complete, by Instrument and Collection
(minutes)

	First Collection		**Second Collection**		**Third Collection**	
	Sort	Importance	Sort	Importance	Sort	Importance
mean	0:21:48	0:04:31	0:16:02	0:03:25	0:11:25	0:03:14
sd	0:07:15	0:01:53	0:07:45	0:01:02	0:04:49	0:01:07

In Table 2 mean times (for all respondents) to sort and scale between 39 and 46 cards are displayed by administration, as are the standard deviations of each. Noting the reductions in completion times, three mechanisms may be at work: respondents became more familiar with the software and administration process, respondents became more domain-expert and required less time to make choices, and respondents remembered previous responses and reproduced them. Most likely, all mechanisms contributed to the reduction, but it is possible to make observations about each.

Regarding the possibility of familiarity with SortDesk and the administration process, we note that the "point-click-drag-drop" methods of SortDesk are common to many current software packages and that these respondents were all apparently computer literate. If completion times shortened because of familiarity with the software it did not amount to much and most of that probably occurred in the first administration, between the first sort task and the final Difficulty scaling.

A more interesting question, as to whether cultural competence rose and thus reduced required response time, cannot be conclusively answered with regard to pile sorting. Unfortunately, at least one of the assumptions of Cultural Consensus Theory (see Chapters 2 and 5) was violated in the pile sorts. While the Theory does not permit unequivocal assignment of violation, that respondents might have (violating Assumption Two) is less likely because administrations were individual and because the Importance and Difficulty portions typically did meet the assumptions of Cultural Consensus Theory. In other words, respondents would have had to collaborate on the first task of each administration and then not collaborate on the succeeding two tasks, an unlikely scenario. Another possibility is that cultural knowledge about the domain was not distributed by competence (Assumption Three) - that certain individuals had more correct views of how sub-domains of the elements should be clustered in terms of similarity and those sub-domains did not intersect much. However, there is no obvious basis for choosing this explanation either, as these were teams of individuals without prior experience in the domain – presumably respondents were evenly naïve.

It seems most likely that the unstructured sorting task failed to tap enough inherent consensual cognitive structure in the team members i.e., that Assumption One, the existence of a coherent culture, may not have obtained. After all, these were teams composed largely of strangers, attempting to structure their work in a highly ambiguous context. This explanation fits with the fact that Cultural Consensus Theory did hold in

(most instances of) the cognitively more structured Importance and Difficulty tasks. That is, providing domain dimensions *a priori* helped respondents structure the domain and those personal structures had sufficient overlap to constitute a culture and avoid violation of Assumption One in the two scaling tasks. In any event, we lack a firm basis for deciding if respondents became more culturally competent in assigning elements to similarity clusters in the pile sort task.

Still, the average time to complete the sorting task in the third administration is almost half that of the first, a marked reduction. Interestingly, the standard deviation, as a fraction of time to complete, rises across the administrations, suggesting that while everyone got faster, some got much faster and others did not. In view all of these circumstances, we suspect that much of the reduction in time to complete the sorting task stemmed from a learning effect – that respondents recalled previous sort clusters, some respondents more proficiently than others, and reproduced them. For collective similarity competence (pile sort competence) to have risen sufficiently to produce such a dramatic reduction in response time, a cultural consensus regarding similarity would have had to existed – at least by the third administration – and we see that it likely never existed because the pile sort singular value decompositions (SVDs) all failed the tests of Cultural Consensus Theory.

The scaling task shown (Importance) also took less time in succeeding administrations. In this case, however, response time reduction cannot solely be due to learning to scale things the same way each time because respondents changed their scalings, as described in the next section of this chapter. Further (see Figure 18, Chapter 2), the inherent lack of visible structure on screen at the end of either scaling task makes memorization exceedingly difficult and therefore makes reproduction from memory on subsequent scalings highly problematic for respondents.

A more plausible explanation for response time reduction is a combination of facility with the software and actual collective competence increase. Recall that Importance and Difficulty scalings consistently met the criteria of Cultural Consensus Theory, bear in mind that inter-respondent mean correlation rose in the scaling tasks most dramatically between first and second administrations (see next the section of this chapter), and note the pattern of reduction in time to complete: a large reduction between first and second administrations and a slight reduction between the second and third. There is also a corresponding drop in the standard deviation as a fraction of mean time to complete, more evidence of a rise in collective competence due to shared culture.

Returning to gauging respondent burden, we note the final result that gives insight into what burdens respondents, the rate at which pile rationales ("titles") were entered during the sorting. For all administrations across all three major groups of respondents, title entry was consistently high: very nearly all piles were titled. The effort expended to do this may not have been great, however, because titles were often very terse, particularly in later administrations. Interestingly, most of these were entered manually, using the respondents' own words rather than the automatic entry (click a card) option. Opportunity for semantic analysis of these data remains and might yield more insight into how respondents structured the domain.

THE DISCRIMINATION ANALYSES

CONSENSUS ANALYSIS

Consensus analysis takes as input a respondent by domain element magnitude scaled matrix in our application. The consensus analysis routine found in Anthropac 4.0, a commonly used anthropological analysis software package, first computes a between-respondent correlation matrix. (Borgatti 1992) Then, singular value decomposition (SVD) is done on that correlation matrix. If the resultant vector (the putative

competences of the respondents) is all-positive and the first eigenvalue is at least three times greater than the second, then one may assert that the cultural consensus model holds. (Romney 1986)

Under the cultural consensus model, we assume that those with most perspective centrality know most. With the correlation matrix, we can rate levels of knowledge as a function of respondent centrality. Using those levels to weight an average for each domain element's valence, we estimate the "cultural truth" in the form an answer key. Our analysis was done under the assumption of magnitude scaling by the mechanisms of SortDesk. That is, we assume that all collected distances are discriminated by better than one part in 10,000, and respondents use SortDesk's display features to visually rationalize their scaling, thus affording a reasonable approximation of magnitude scaling.

Consider an example of application Cultural Consensus Theory to data from the Summer 1997 Special Project. In this case we have taken both importance and difficulty scaling data from one of the five teams, produced respondent by domain element matrices for each of the three administrations, and generated inter-respondent correlation matrices. On doing Singular Value Decomposition of these correlation matrices, we found that all six (2 dimensions by 3 administrations) of the scalings had positive eigenvectors. Four of the six scalings (all 3 Importance and one of the Difficulty) passed the eigenvalue ratio test and the remaining two very nearly passed. It is perhaps more interesting that two of the discriminations failed the test despite having had explicit, presumably salient dimensions (Importance or Difficulty) *a priori*.

Table 3 shows between-respondent correlation matrices (data shown by administration) from the Importance scalings. Note that in the third administration, three of the respondents failed to complete the task. The first administration resulted in a slightly negative mean between-respondent correlation, indicative of neither very much consensus nor dissensus - team members neither agreed nor disagreed about the relative importance

of various project activities. Given the intentionally vague overall objectives set by the organizers of the Summer 1997 Special Project, this seems to make sense: team members would have little basis for agreeing or disagreeing.

Table 3: Activity Importance, Between-Respondent Correlations and Team Means, by Administration

	-0.012 Team mean			First Administration			
Member 1	1						
Member 2	-0.100	1					
Member 3	0.035	-0.396	1				
Member 4	-0.091	0.144	-0.500	1			
Member 5	-0.159	0.196	-0.561	0.391	1		
Member 6	-0.307	0.353	-0.452	0.320	0.333	1	
Member 7	-0.339	0.405	-0.420	0.150	0.365	0.379	1

	0.306 Team mean			Second Administration			
Member 1	1						
Member 2	0.458	1					
Member 3	0.327	0.056	1				
Member 4	0.229	0.035	0.321	1			
Member 5	0.658	0.279	0.493	0.237	1		
Member 6	0.464	0.315	-0.007	0.183	0.371	1	
Member 7	0.533	0.061	0.329	0.169	0.516	0.390	1

	0.307 Team mean			Third Administration			
Member 1	1						
Member 2	-	-					
Member 3	0.580	-	1				
Member 4	-	-	-	-			
Member 5	0.137	-	0.283	-	1		
Member 6	-	-	-	-	-	-	
Member 7	0.223	-	0.181	-	0.437	-	1

The second administration resulted in a larger (more positive) mean correlation, suggestive of condensation in perspective or increase in consensus. This may be explained

by noting that at the outset team members expressed much concern and dissatisfaction about the lack of direction given them and organizers noted that most team members were undergraduate students chosen for achievement in (relatively well structured) academic circumstances. (Monahan 1997) To deal with the discomfort stemming from lack of goal clarity, this team may have generated a tolerable internal sense of relative importance of project accomplishments in the period between the first two administrations. The third administration has fewer respondents (three failed to complete this administration) and thus less potential variation, so its similarity with the second may be factitious. In any event the mean correlation remains positive but not high, indicating that team members continue to disagree about project activity relative importance.

Turning attention to Table 4, we note first that some of the variance in between-respondent correlation is due to team members who changed their minds. In Table 4 data are sorted from left to right in order of decreasing self-consistency, indicating that Member 6 most maintained the same evaluation of relative importance and Member 2

Table 4: Respondent Self-Consistency and Mean Agreement with Rest of Team, by Administration

	Self-Consistency Across Administrations							
	0.777	0.660	0.519	0.477	0.470	0.462	0.179	
	Member 6	Member 3	Member 4	Member 7	Member 5	Member 1	Member 2	SD
1	0.104	-0.382	0.069	0.090	0.094	-0.160	0.101	0.175
2	0.286	0.253	0.196	0.333	0.426	0.445	0.201	0.093
3	N/A	0.348	N/A	0.280	0.286	0.313	N/A	0.027
	Mean Agreement with Rest of Team by Administration							

most changed his mind. Interestingly, neither Member 6 nor Member 2 completed the third administration, lending credence to the assumption that simple error and sloppiness did not tend to lower self-consistency the more administrations a respondent completed.

Below is a by respondent tabular narrative that attempts to cast the statistics of Table 4 in a consensus-oriented perspective. It is intended to be thought provoking, not definitive.

Member 1	Member 1 began in mild disagreement with the rest of the team, found herself situated centrally by the second administration (had a perspective on importance that was most in agreement with the other team members), and finished with a somewhat less central perspective that was nevertheless the second most central final perspective. Her overall self-consistency was middling to low, indicating that she changed her perspective in order to arrive at greater centrality.
Member 2	Member 2 began in mild agreement with the rest of the team, became a bit more central on the second administration, and failed to complete the third. His overall self-consistency was by a large margin the lowest, indicating that he had to change perspective radically in order to join the team consensus. Because he began with some agreement and yet had to change dramatically to join consensus, we may conclude that the center of team consensus moved between the first and second administrations.
Member 3	Member 3 had the greatest self-consistency of those team members who completed all three administrations. She began with substantial disagreement with the rest of the team and yet had an increasingly central perspective. We must conclude that the team came around to her perspective that, by the third administration, was most central.
Member 4	Member 4 began with mild agreement, moved slightly toward the (new) team center on the second administration, and failed to complete the third. He has a high-middling self-consistency, indicating that his first administration perspective may have been close to the second administration center.
Member 5	Member 5 began with mild agreement, found herself near the center of perspective on the second administration, and moved peripherally on the third administration. Her self-consistency was middling, raising the question as to what caused her to deviate from the consensus at the end of the project.
Member 6	Member 6 had the greatest self-consistency and independence. He, too, began with mild agreement to a fuzzy consensus and improved his centrality without, apparently, changing his mind very much. From this we may conclude that the team moved somewhat closer to him. It is a pity that we lack a third response from which we might see if his independence and consistency continued.
Member 7	Member 7 had middling self-consistency, low initial agreement, a sharp rise in agreement at the second administration and a decline at the third. The declines at the third administration raise the question of what may have caused the divergence.

Finally, we note that the standard deviation of member mean agreement with the rest of the team falls steadily across the three administrations (Table 4). That is, even though team members appeared to agree less about relative importance in the third administration

than in the second, they had a consistent amount of disagreement. The team had perhaps settled to "agreeing to disagree" - it consisted of individuals with a modicum of perspective overlap rather than a central clique surrounded by independent outliers.

Other inferences, of interest to product development practitioners, can be made by inspection of estimated true magnitude scalings and knowledge levels. Putatively knowledgeable people may be given influential roles. Project activities with high Importance and low Difficulty may be addressed directly (see Figure 20). Project activities with high Importance and Difficulty and high variance across respondents may be identified and used to focus team dialog work. In Figure 21 is a bubble plot of the estimated true magnitude scalings for Importance and Difficulty for the ASC Derivative Product Development, by domain element. By bubble plotting true Importance or Difficulty magnitudes (the "answer keys") on the same axes and sizing bubbles so that their areas are proportional to team variances in domain element scalings, it is possible to identify domain elements of high importance or difficulty and high dissensus in scaling across the team. The team can focus dialog work to strengthen consensus by selecting items of importance and difficulty about which there is much apparent disagreement and avoid wasting time where consensus is already strong.

In Figure 22, the answer keys for each administration of Importance scaling for a 1997 Summer Special Project team are plotted with the grand overall Importance answer key that resulted from simultaneous consensus analysis of all three of the team's administrations of Importance scalings. In this plot, the farther a point is from the center the greater its Importance. Noting, for example, the axis at the right of center at the bottom titled "deal with lack of computer availability," it might be inferred that (un)availability of computers became an increasingly important problem as the project progressed. In fact, many Project participants expressed increasing frustration with lack of computer access as the Project went on. (Monahan 1997) At the upper right, the axis titled "satisfy our customers" shows that team members began by thinking that this was

most important and that, by the end of the Project, had decided that it didn't really matter that much. This, too, is consistent with a sense of frustration as to exactly what "satisfying an (ill-defined) customer" constituted, about which they also complained. (Monahan 1997)

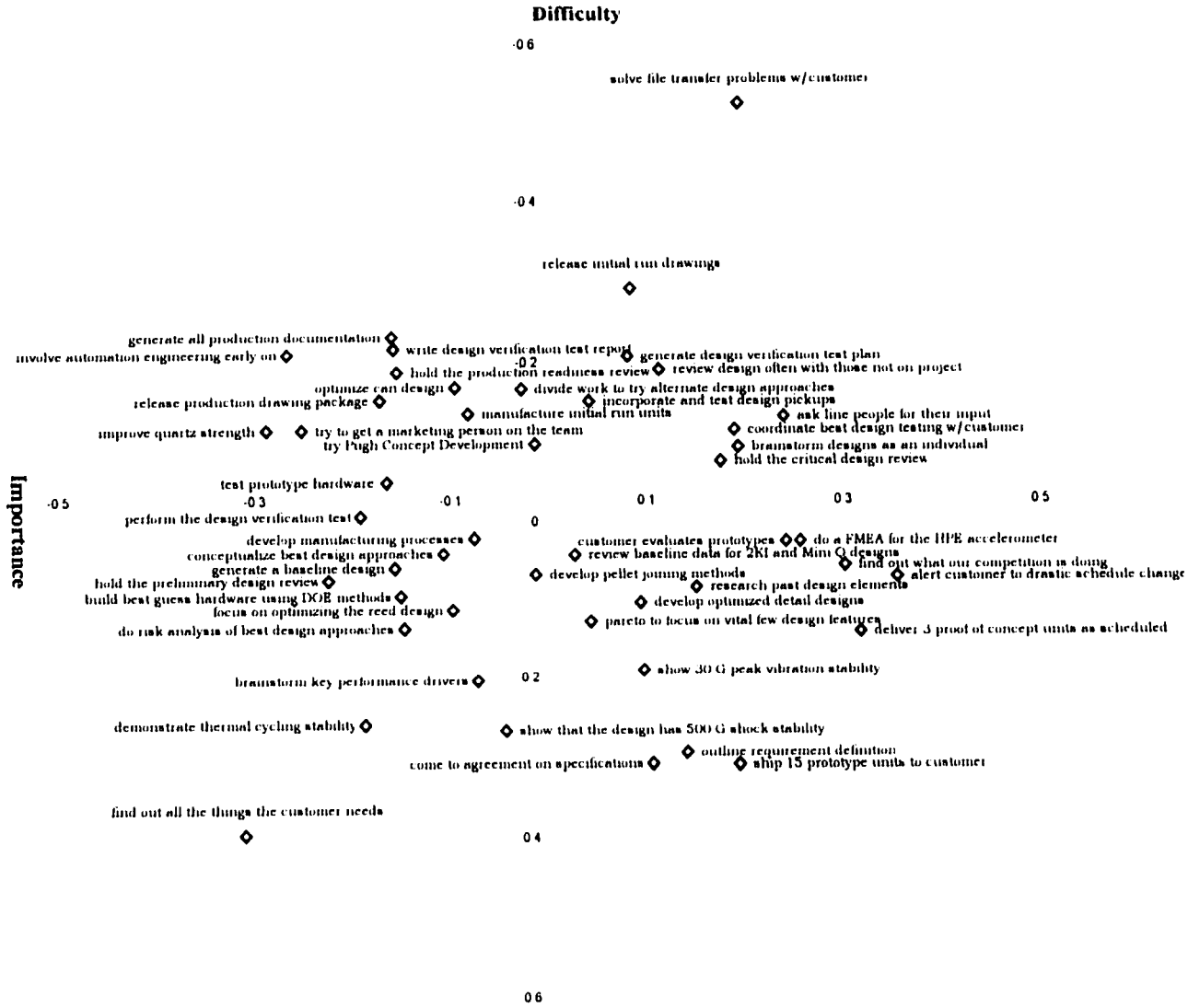


Figure 20 : ASC Importance vs. Difficulty

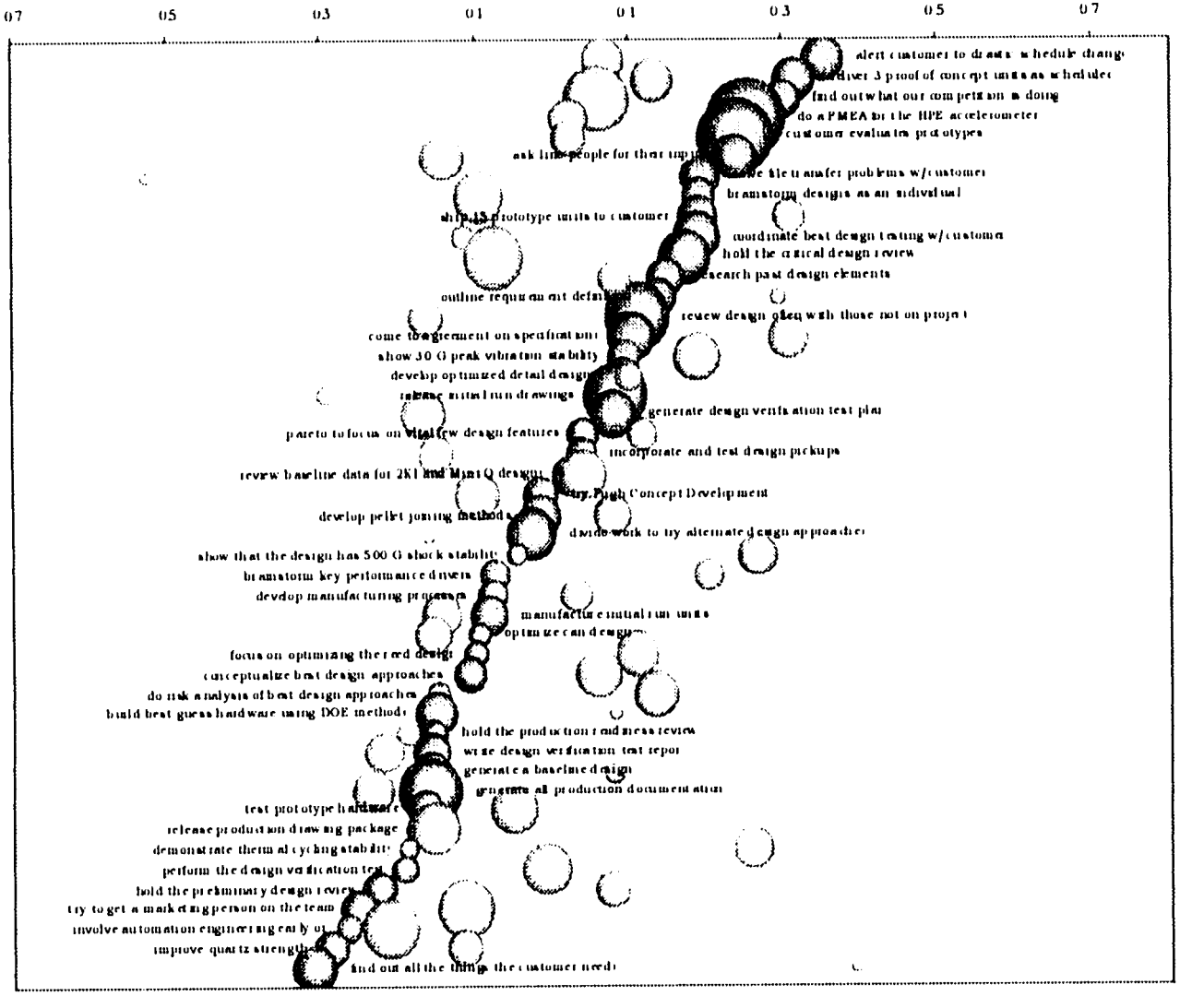


Figure 21 : ASC Importance and Difficulty Keys with RMS Deviations

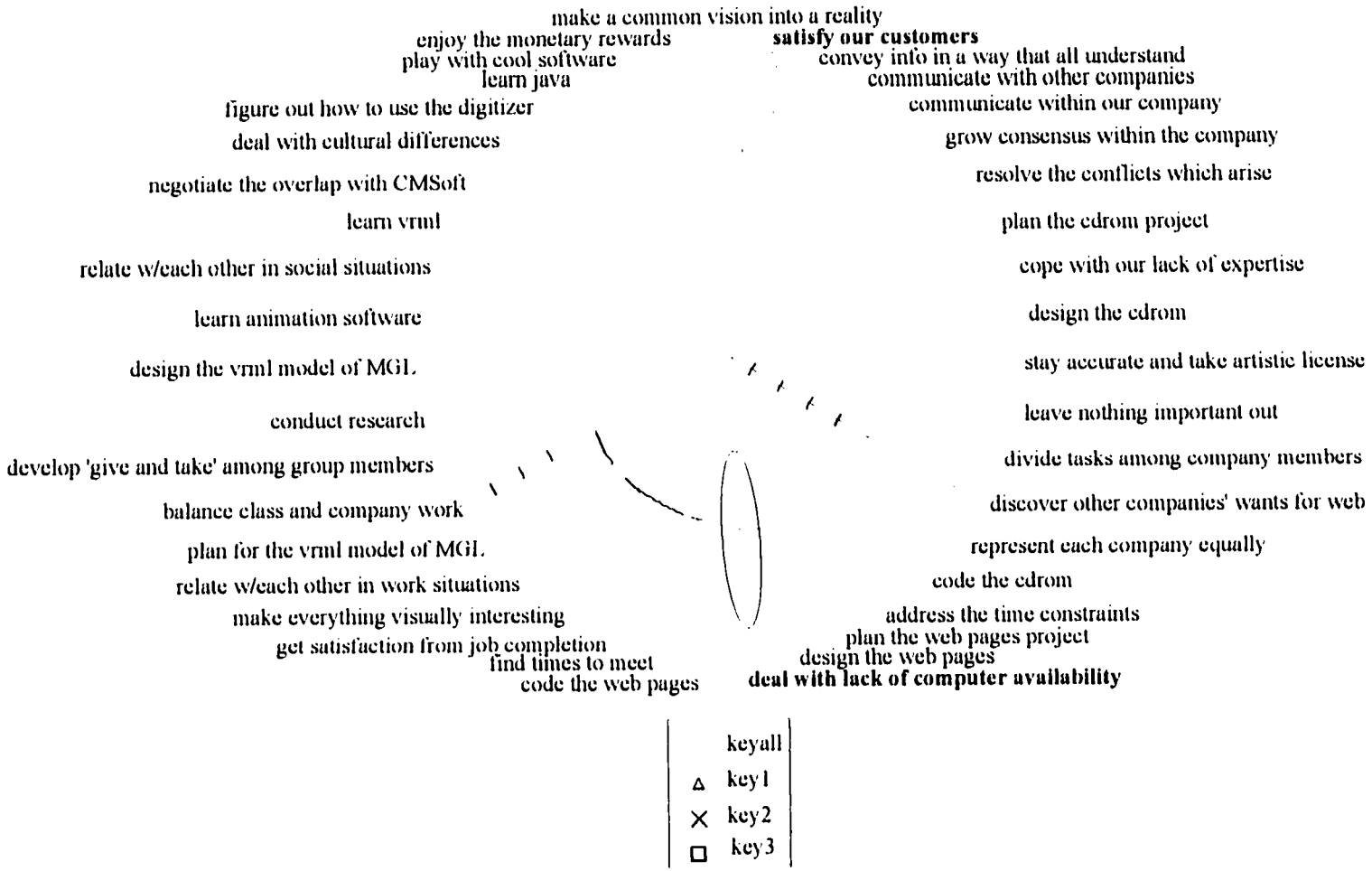
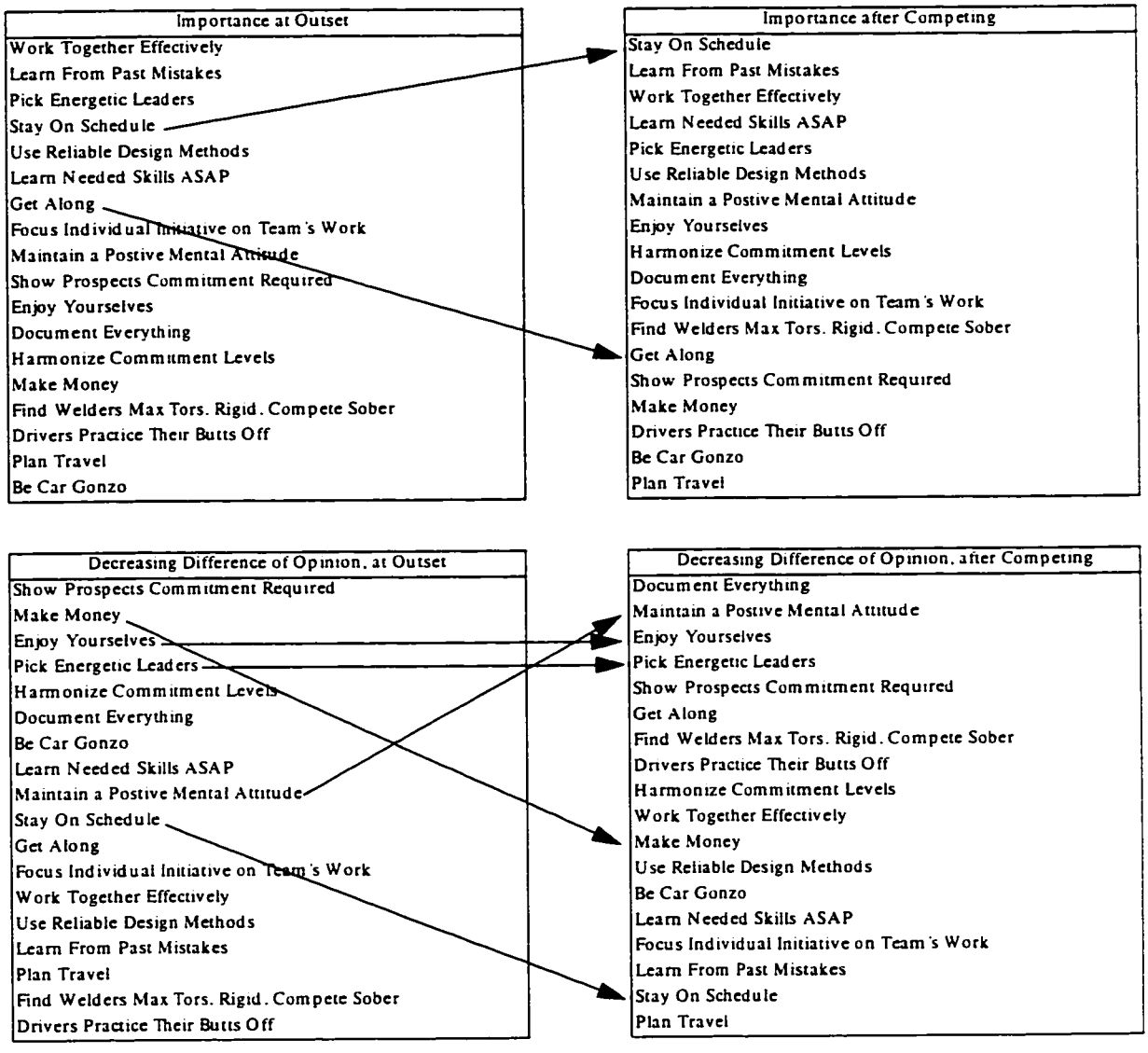


Figure 22 : A Summer 1997 Special Project Team's Importance Keys Comparison

Analyses may also be displayed (see Table 5) as tables of domain elements sorted by magnitude or dispersion and compared diachronically, perhaps before and after a presumably significant project event. In this instance, the tables at the top show that SAE Formula Car '97 team members appear to have changed their minds about the relative Importance of "Stay on Schedule" after their rather painful experience (a very poor showing) at the national competition. Further, we note that after competing they also had less variation in opinion of the Importance of "Stay on Schedule:" in the table on the bottom left that domain element has a middling mean squared deviation, whereas in the lower right table it has second to least. It seems as though they have reached greater consensus about the necessity to "Stay on Schedule." The Importance of "Get Along" dropped precipitously between administrations. High variation (dissensus) is found in both administrations regarding the Importance of "Enjoy Yourselves" and "Pick Energetic Leaders," dissensus about the Importance of "Maintain a Positive Mental Attitude" rises most sharply in the second administration, and the sharpest increase in consensus occurs regarding the Importance of "Make Money." Referring back to the tables above, we note that in each of them "Make Money" has little importance: apparently a few had thought it important at the outset but by the second administration consensus about its relative unimportance had risen.

To conclude the analysis based on Cultural Consensus Theory, consider Table 6. In it a Summer 1997 Special Project Team's Importance scaling competences is shown. Strikingly, the mean cultural competence of the team hardly varies at all but the competences of individuals swing widely. It is hard to see how this might happen unless the locus of the center of consensus actually moved between administrations, i.e., that perspectives shifted. It would be interesting to compare the relative competences with measures of team leadership and networks of communication.

Table 5: Formula Car '97 Average Importance Scalings and MSD Before and After Competing



Presumably, the team would have had greatest success if Member 3's perspective had been (1) actively adopted by all, thus yielding higher consensus, and (2) because she had highest mean competence, her view was the most correct view the team had the capability to generate.

Table 6: A Summer 1997 Special Project Team's Importance Scaling Competences, by Administration

	1	2	3	individual means
Member 3	78.9%	44.2%	75.1%	66.1%
Member 5	62.5%	81.3%	41.6%	61.8%
Member 1	27.5%	85.8%	64.8%	59.4%
Member 6	62.7%	50.3%		56.5%
Member 7	59.2%	62.3%	39.9%	53.8%
Member 2	47.9%	36.5%		42.2%
Member 4	49.6%	31.3%		40.4%
	55.5%	56.0%	55.4%	
	administration means			

MULTIDIMENSIONAL SCALING

Multidimensional Scaling (MDS) attempts to fit into (typically) two dimensions a map of n domain elements in which every pair of elements is separated on the plane by a distance proportional to the distance between them in the n-1 space of explicit pairwise similarities. Such a projection will usually force some or all of the pairs into a configuration with inter-element distance error. A measure of this error (termed stress) is used to assess the validity of assuming that the map's axes represent underlying dimensions within the cognitive domain. When stress is under 0.15 it is deemed acceptable within the community of cognitive anthropologists. (Borgatti 1992) If stress is acceptable, making inferences about the semantic nature of each axes' gradients is considered defendable. It is best, of course, for stress to be as low as possible and to test any inferences explicitly after detecting evidence for them.

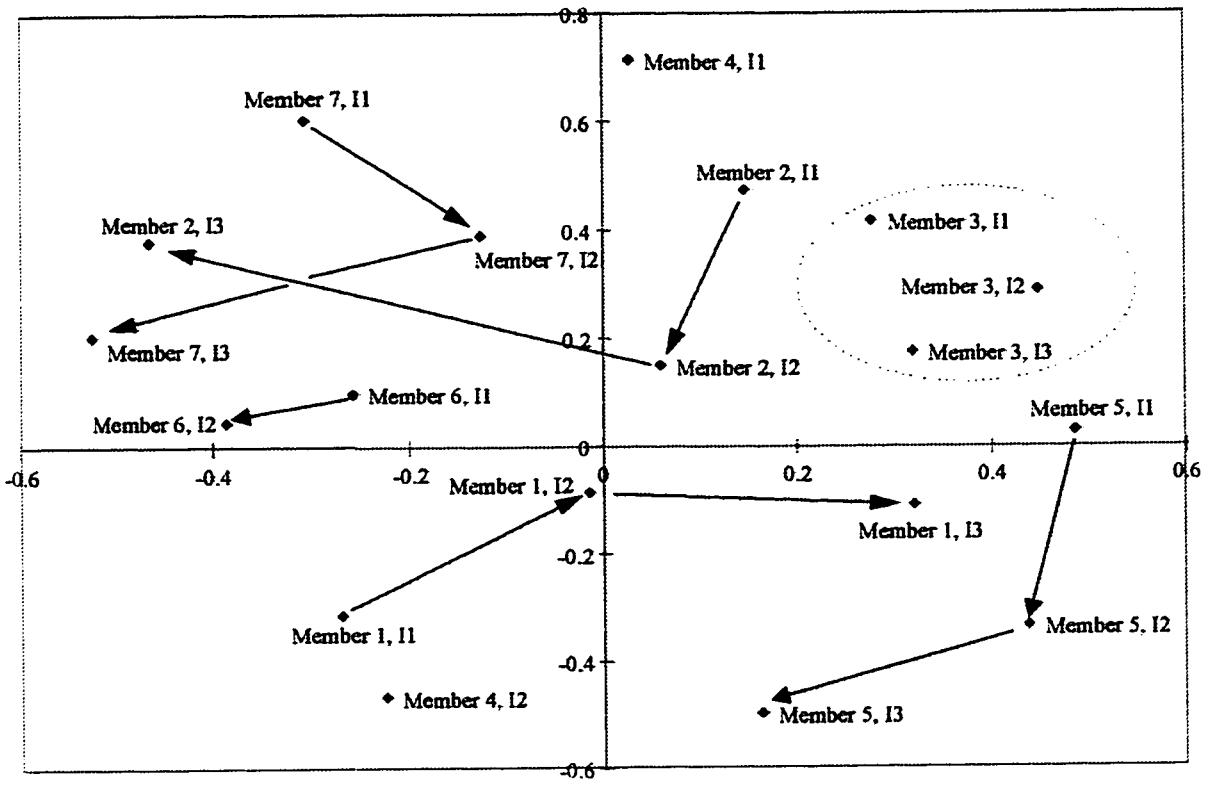


Figure 20: A Summer 1997 Special Project
Team's Importance Correlation Map

The clustering of elements in the plane may be considered meaningful, too. In Figure 23 above, the overall between-respondent Importance correlation matrix for a Summer 1997 Special Project team is projected into a two dimensioned map using Multidimensional Scaling. Each respondent is treated as three respondents, one for each of three administrations. It is important to note that the projection fails, but just slightly, the acceptance criterion: residual stress is computed at 0.18. If we take the liberty to interpret this chart not for evidence of underlying dimensions in team cognition, but for evidence of consensus and movement toward consensus, we may make the observations of the following sort. It appears that respondent Member 4 moves dramatically from one pole of perception to the opposite between the two administrations for which we have his data.

Member 1, Member 2, and Member 7 begin at relatively outlying positions in the cognitive domain, achieve significant centrality in the second administration, and then move back to outlying positions for the third. We also note the relative consistency of Member 3, who remains in the same region of the perception space throughout the administrations. Evidence of Member 3's consistency is also found in Tables 3 and 4 above, and will also be apparent in the subsection below. Indeed, all of these descriptions neatly match those in Table 4.

CORRESPONDENCE ANALYSIS

Correspondence Analysis, as recently consolidated by Nerlove and Romney (Nerlove 1993), encompasses a variety of eigenstructure based methods used in anthropology and allied fields to draw inferences about interrelated variables. Application examples include using correspondences between archeological artifacts to construct an interval scaled time line of tool usage (paleoPD!), and exploring attitudes about contraception to find common, underlying views and to discover differences in views between men and women.

For us to take full advantage of correspondence analysis we would need to elicit more and different data. If researchers can use our work to further investigate the **Clausing Conjecture** (see Chapter 2, section 2) and have the resources to include estimation differences and changes in consensus, motivation, expertise, resourcing, schedule fidelity, and product market value, then correspondence analysis may aid them. However, because we have data with underlying dimensions *a priori* (i.e., Importance and Difficulty) we do not need correspondence analysis to identify them. Accordingly we may use display methods that are used in correspondence analysis after underlying dimensions (often, two) are identified.

In Figure 24, plotted as an X-Y scatter plot, are the Importance and Difficulty magnitude scalings for one respondent, by domain element, by administration

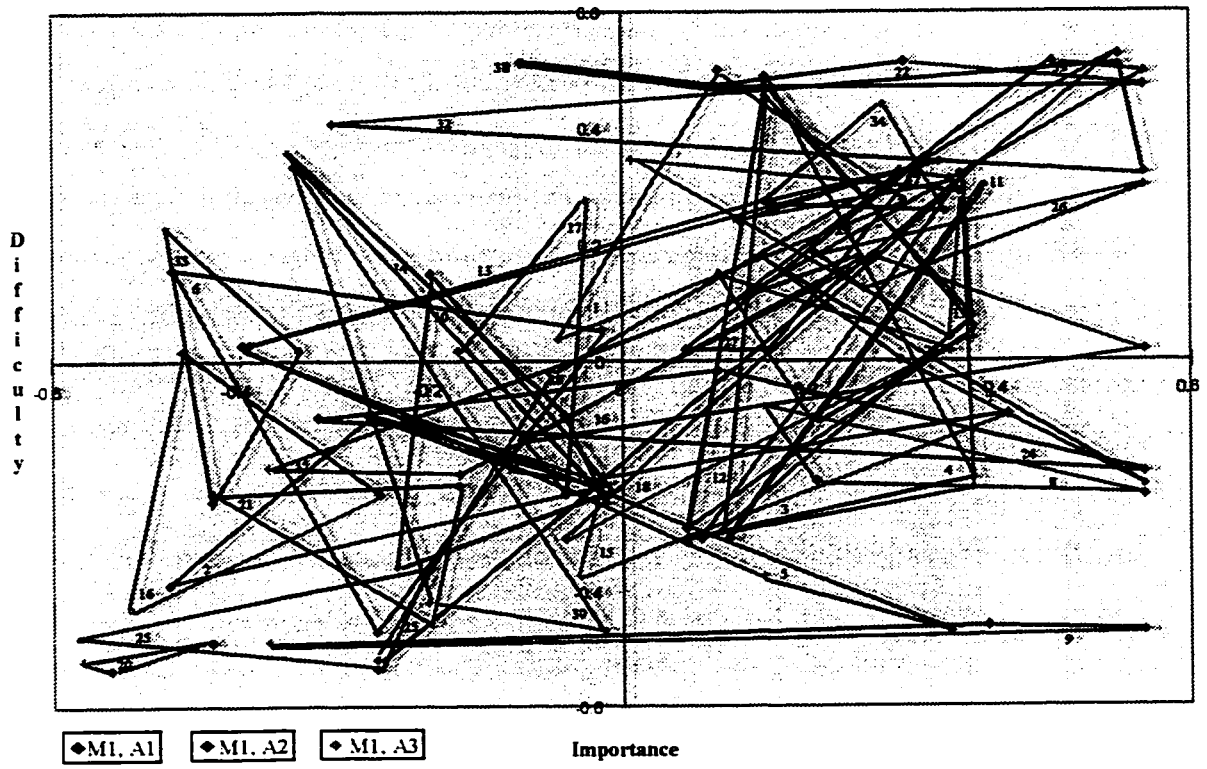


Figure 21: Member 1's Importance and Difficulty

Lines, forming a triangle have connected each domain element's three points (one for each administration). The size of a triangle, then, indicates how much the perspective of the respondent has changed across the administrations. Smaller, more distributed triangles imply consistency in perspective and precision in distinction.

Compare this respondent's plot with a teammate's, in Figure 25. For the second respondent, triangles are much smaller and overlap less, indicating a more consistent, more discriminating perspective across the administrations. Recall that Member 3 also has highest mean competence. (See Table 6)

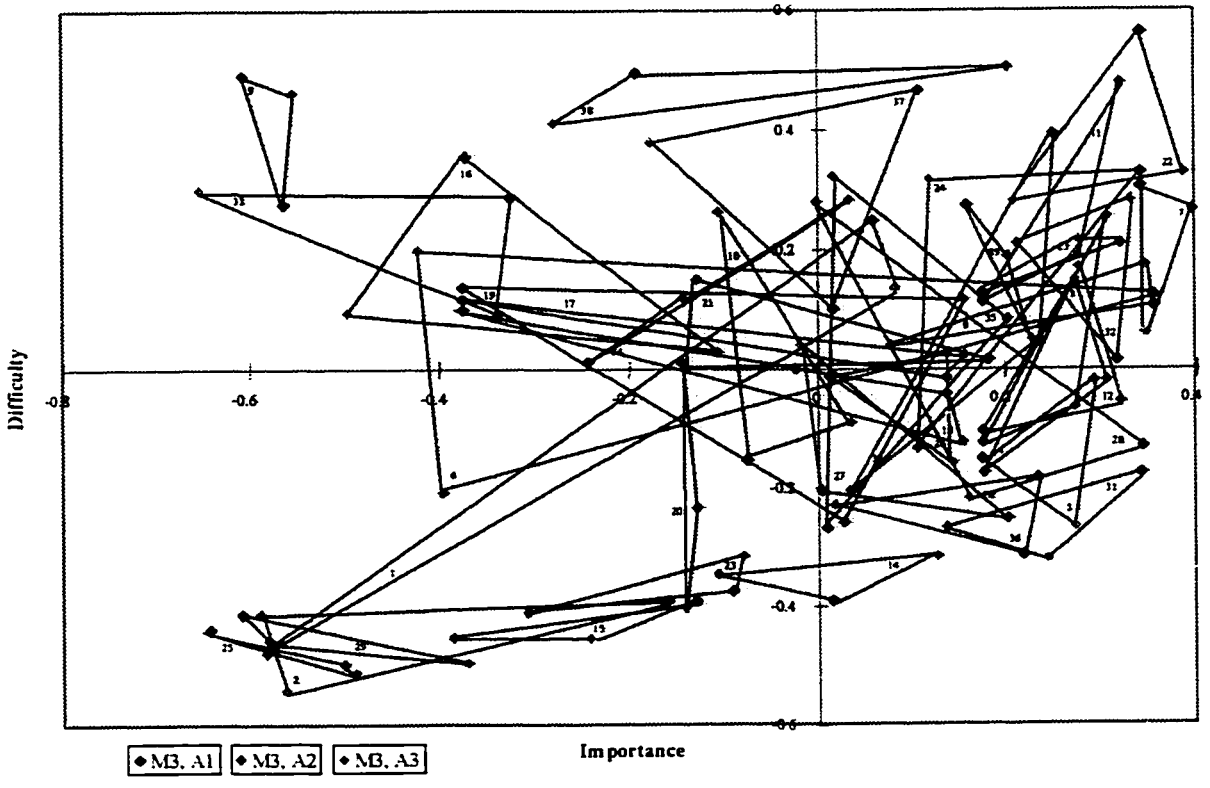


Figure 22: Member 3's Importance and Difficulty

CHAPTER 5: CONCLUSIONS

INTRODUCTION

This chapter has two sections in addition to this introduction. The next section addresses questions of respondent burden and the final section draws conclusions regarding the use of various methods for the characterization of consensus in product development teams.

METHOD BURDEN

To introduce the discussion of the question of respondent burden, let us note that perceived burden does not rest solely on the difficulty of the task. It is, in fact, in the balance between respondents' level of motivation and the difficulty of the task that drives compliance, which is finally the salient issue. Below are our conclusions regarding the issues raised in the Introduction and Problem Statement chapter of this dissertation. In summary, we claim that the sorting and scaling tasks are simple and quick but we cannot say absolutely that the burdens are acceptable, only that the tasks are not terribly time consuming. We expect that effort to motivate respondents will probably work better to stimulate compliance than attempting to significantly lower the (already low) burden.

HOW MUCH BURDEN WILL PEOPLE PUT UP WITH?

As we see in Chapter 5, for domains with roughly 40 elements, it is reasonable to expect that an average respondent will take about 20 minutes to complete her first pile sort and that her third sort will take about half that much time. Each magnitude scaling of that many elements will take about a quarter of the time that the sort takes and also takes less time with subsequent administrations. Asking for pile rationale turns out not to be the burden feared, apparently, except in the instance of the "full Boster" sort (to unity and

singletons). In the single free sorts, we received fair compliance SAE '97 administrations and broad compliance in the others. Following the caveat of the introduction of this section, SAE '97 compliance is most likely a function of uneven motivation between respondents: those who responded regularly tended to complete requests for rationale; those who responded only once (as many did) tended to skip entry of rationale.

Domains with twice the number of elements tended to take twice as long to sort or scale, staying roughly proportionate as respondents' learning effects reduced time per administration over multiple administrations. While this may seem sensible, in actuality a respondent must accomplish approximately four times the number of (pairwise) discriminations when a domain has twice the number of elements. If it takes only twice as long to discriminate domains with twice the elements, greater domain elaboration may be discriminated more efficiently than simpler elaboration, at least within the range of domain element counts feasible for legibility on SVGA PC screens. Respondents' impatience with triads tests makes sense in light of this apparent discrimination efficiency. Perhaps there is some perceptual/cognitive synergy in considering larger sets.

WHICH PRESENTATIONS SEEM MOST BURDENSOME?

Respondents complained about triads tests most. With domains of the sizes we contemplate (ca. 40 elements), triads tests are probably not feasible. Perhaps this ought to be unsurprising because even a highly fractionated lambda 1 balanced incomplete block experimental design for a 25 element domain requires respondents to make about 200 triad discriminations. While each presentation may take less than 5 seconds, 200 presentations can run to 15 minutes or more (an apparent eternity when bored) of rather mindless activity.

How can a Graphical User Interface (GUI) help? Hurt?

Any software embodiment of the tools tends to remove the necessity that researcher and respondent be simultaneously present, thus freeing respondents to choose when they wish to do the tasks. Software tools also collect data flawlessly and in electronic formats, which are easy to manipulate. Software tools make a uniform presentation to respondents that can be carefully tailored to minimize response contamination.

Apart from the general advantages of a software embodiment, using our graphical user interface had the advantage of affording respondents use of a novel medium with a look and feel which mimicked physical cards – something that went a long way to making the interface operate in an intuitive fashion. The intuitively obvious nature of the GUI made large amounts of instruction in software use unnecessary. With a proper tutorial and mistake proofing, it seems entirely possible that respondents would never need to meet the researcher.

The snap-join and pile tiling features make the cards easier to arrange, move, and position than physical cards, which tend to shift position and obstruct each other, and which must be individually grasped to be moved.

Magnitude scaling of domain elements, very awkward with physical methods (the best compromises here being the assignment of numbers to scale weights and the drawing of lines whose lengths must be measured by researchers; see Lodge 1986), is straightforward with our graphical user interface.

The principal drawback to software embodiments is lack of researcher presence. When we administered instruments in person, we had greater compliance. Concerning the graphical user interface, the principal drawback comes from the size restrictions of the computer screen. Cramped screens reduce the legibility of cards, the lengths of statements

on cards, and lead to cluttered sort spaces. There are ways to ameliorate these problems, some of which we address in Chapter 7.

WHAT SORT OF MISTAKE PROOFING CAN WE DO?

SortDesk incorporates a number of mistake-proofing measures. They generally fall into two categories: helping respondents avoid misplacement of cards, and keeping respondents from inadvertently aborting the administration or mistakenly advancing to a next segment before completing the current one.

At startup SortDesk seizes the entire screen and prevents display of taskbars. Apart from explicitly offered opportunities to exit SortDesk, respondents may quit only by turning off the computer or by pressing the control, alt, and delete keys simultaneously to take control at the operating system level. Although our respondents can be considered computer literate, few of them knew how to do this, effectively locking them in until task completion.

During the free sort the Done button is disabled if the criteria in the first instruction task box (more than one pile, fewer than singletons) are not met. The Done button is also disabled during the entry of titles for piles and until all of the cards have been dragged off the deck during the two scaling tasks.

The drag icon changes to the universal "don't do this," circle-with-a-slash icon if the respondent drags a card or pile over an instruction task box, a command button (e.g., Done), the title entry text box, or off the SortDesk. If the object is dropped while that icon displays, the drag is nullified and the screen returned to the pre-drag state. Visually it appears that the dragged object jumps back to where it had been.

For data collection assurance, SortDesk will not run if no diskette is in the A: drive. All "writes" to diskette are in the "Append" mode, thus forestalling inadvertent overwriting of

data. In the data file, a date and time stamp from the operating system signals each administration and segment startup.

The log-in screen presented on initiation of SortDesk (not shown) requests that respondent enter the date and his password. Date entry provides a check of the system clock setting. Password protection gives respondents a sense of privacy and prevents accidental use of another respondent's diskette (each has his own).

CHARACTERIZING CONSENSUS

On the basis of Cultural Consensus Theory and less formal criteria, we believe we have detected changes in consensus in small teams, most defendably along pre-selected dimensions of interest e.g., Importance. That is, the locus and dispersion data were different for different administrations to the same team, but hypothesis testing based on the normal theory to provide justification for assessing the significance of these differences is lacking and appropriate non-parametric hypothesis tests were not identified. In the next chapter, we discuss future work that would develop a theoretical basis for such hypothesis testing and suggest means to empirically establish the tests. Below are discussions of the issues raised in the Introduction and Problem Statement chapter.

HOW DOES CULTURAL CONSENSUS THEORY APPLY?

The assumptions of Cultural Consensus Theory are: (Romney 1988)

Assumption One: Common Truth:

> There is a fixed answer key applicable to all respondents. In other words, all respondents come from a common culture, that whatever the team's cultural reality is, it is essentially the same for all of them.

Assumption Two: Local Independence:

> The respondent-item response random variables satisfy conditional independence (conditional on the correct answer key). This means that because each team member responds independently, inter-respondent correlations are solely a function of concordance between individuals' appreciation of cultural truth.

Assumption Three: Monotonicity of Items.

> Respondents who have more competence an any subset of questions will have more competence on all subsets.

In the instance of the Summer 1997 Special Project team analyzed in Chapter 4, we used the Anthropac software (Borgatti 1992) to construct respondent by respondent similarity matrices of judgment regarding Importance and Difficulty of project aspects. Anthropac then used a minimum residual method of factor analysis to estimate each respondent's competence. In all cases analyzed, these competences (comprising the eigenvector) were all positive and the ratio between the first and second eigenvalues of the factor analysis was greater than three. If any of the assumptions of Cultural Consensus Theory had been seriously violated this would not have occurred. (Romney 1988) Thus, we assert that enough inter-respondent perspective consistency existed to form a coherent collective cognitive domain regarding their product development. This is a bit remarkable, because Cultural Consensus Theory was developed to investigate much more basic, slowly changing, and widespread cognitions in established cultures, and these team members began as strangers with little or no experience in product development and tried to hit a moving, vague target.

CAN WE DISCRIMINATE PERSPECTIVE VARIANCE?

We assert the following:

1. Consensus existed for a Summer 1997 Special Project team on each administration of Importance magnitude scaling.

2. Consensus for that team must have changed between administrations because, despite virtually unchanging team mean competence, individual respondents had widely varying competences across administrations (see Table 6, Chapter 4). That is, it is therefore fair to conclude that if a consensus existed on each administration, it must have changed between them, unless respondents made well-matched offsetting moves in perspective. Put another way, although team mean centrality remained constant, individuals' centralities changed - they moved in and out of the center of consensus. For this to occur without change of consensus center, team members would have had to move in a neatly offsetting fashion, a substantially unlikely occurrence given complexity of the scaling tasks (39 elements), the by-individual administrations, and the demonstrated validity of the independence assumption.

3. For that team the amount of consensus increased as the Summer 1997 Special Project progressed, most clearly between the first and second administration (the third administration includes only four of the seven team members' responses). Referring to Table 3, Chapter 4, we note that the mean correlation between team members moved from a slightly negative value to a substantially positive (though not a high) value between the first and second administrations. This is evidence of higher consensus, but not an overwhelmingly great one. Such is entirely consistent with the confusion regarding objectives expressed by participants in the 1997 Summer Special Project. (Monahan 1997)

Instead of pre-selecting dimensions for scaling (Importance and Difficulty), it would have been preferable to elicit dimensions from the pile sorts via multidimensional scaling. However, for pile sorting to have provided enough sensitivity we would have either needed teams with higher initial consensus about domain perspectives (higher competences) or much larger teams. In fact, the sample size heuristic for pile sorting in unknown cultures is to begin with a minimum of 30 respondents. (Borgatti 1992) This

could have been accomplished if more aspects of SAE Formula Car '97 and Summer 1997 Special Project had been explored across all respondents in addition to teams.

Finally, we note evidence of consensus shift in many of the other analyses of Chapter 4, noting particularly the shifts in ranking and changes in amount of disagreement found in Importance ranking. It is hard not to believe that SAE Formula Car '97 respondents actually became more certain about the importance of Staying on Schedule and that they also agreed much more about how (very) important it was.

CAN MULTIDIMENSIONAL SCALING OFFER INSIGHTS?

The purpose of multidimensional scaling (MDS) is to display a visual representation of the pattern of similarities or distances among a set of objects. For n objects, exact pairwise distances may be described in no smaller than an $n-1$ dimensioned space. Researchers use MDS to project downward from the $n-1$ space into a two- or three-dimension space as an aid to understanding relationships between objects. If the projection (typically approached as an optimization problem) produces a full set of pairwise distances with an acceptable amount of accumulated projection error, the results meet a critical test for validity. When the criteria for valid use of MDS are met, it is often possible to find suggestive indications as to underlying, tacit dimensions in the domain of interest. A classic example of MDS involves using the average prices of air travel between cities on a continent to display an often surprisingly representative map of the continent's cities – in their proper locations relative to each other. In this case, not even all of the actual pairwise geographic distances could be precisely rendered in a two space (consider altitude), but the approximation is good enough for us to regularly use non-topographic maps.

In understanding consensus in product development teams, MDS might be most useful if the conditions discussed immediately above obtain i.e., that a culture exists and the respondents are a sample of that culture. In that instance, valid use of MDS, to discover

underlying dimensions in team cognition or clusters of cognition that might begin to constitute a taxonomy of cognitive domain elements, could provide a basis for focusing team efforts to reach consensus through dialog. By way of example, the directly constructed 2-D display of Figure 20, chapter 4, showing Importance and Difficulty for one Summer 1997 Special Project team might conceivably have been generated using MDS, had proper conditions obtained. In such a case, the dimensions would be unlabeled and inspection of the orientation of various items and clusters could be done to find possible meaning for these revealed dimensions. In good circumstances, testable inferences regarding Importance and Difficulty as underlying dimensions and the meaning of certain clusters might become apparent.

For us the conditions required for strict applicability were not met, yet we find the graphical display associated with MDS somewhat useful for characterizing team members' diachronic perspective consistency and discrimination capabilities. Please see Figures 24 and 25 in chapter 4 for a discussion of this approach.

HOW MIGHT INFORMATION THEORY AID ANALYSIS?

Here again, with neither the cultural consistency nor large enough sample sizes obtaining in our research, we could not determine conclusively if Burton's information theoretic transform for free sort data represents much benefit. (Burton 1972, Appendix A) We devoted no small amount of time to analyzing data transformed in this fashion, despite the defensibility problem, and were unable to find any striking or suggestive results.

CHAPTER 6: FUTURE WORK

INTRODUCTION

This chapter describes some future research and development opportunities, categorized as either extensions to method, application extensions, or extensions to media. The next chapter section explores possible extensions to ethnoscience arising out of the medium of administration, the section following that discusses possible wider application of SortDesk, and the final section proposes elaboration of SortDesk.

METHOD EXTENSIONS

Computer mediation supports simple but careful data elicitation and affords opportunity to use even more complicated experimental designs. Presumably, some interesting work might be done in adaptive presentation of triads tests and coupling of triads testing with pile sorts. Because the methods we used are burdensome without PC mediation, little work has been done in tracking cognitive shifts over short amounts of time. Further application in diachronic investigation of cognitive domains would likely reveal more options for both administration and analysis.

To assess the significance of changes in data collected diachronically, work could be done to establish reliable hypothesis testing. In order that researchers might have confidence in determining whether differences in data are not likely to have occurred by chance, some theoretical development of the effects of levels of random variation on Cultural Consensus Theory results would be required. In addition, because of the theory's dependence on respondents' competences, empirical testing would be required. We have in mind something like the following: choose a cognitive domain of high consensus (e.g., color perception or kinship), administer to a large sample diachronically, test for consensus at

each administration, and measure the naturally occurring variation in response to estimate random variation. Alternatively, use putatively arbitrary domains (e.g., free association to well known terms) in the same fashion.

Following the lead of Freeman, developer of the MAP software application (Freeman 1994), it would be easy to adapt SortDesk to handle two-way magnitude scaling. As an example, consider asking respondents to arrange cards from left to right according to Importance and bottom to top terms of their Certainty about each Importance assessment. Alternatively, use the actual screen coordinates of cards to construct similarity matrices.

D'Andrade, in discussing reasonable and tested arguments from psychology that help explain the empirical robustness of Cultural Consensus Theory (D'Andrade 1995: 214-215), notes that speed of response is correlated with high competence. One could use SortDesk to track response times for use in estimating respondent competence.

Under the conditions that validate multidimensional scaling, for diachronic administrations one could compute the areas (or volumes) bounded by lines connecting the tips of the domain elements' administration vectors. These might be computed for individuals and for teams when using predetermined dimensions such as Importance (showing consistency *vs.* change of mind) in individuals, in teams over time (showing consensus/dissensus changes) or between teams (showing relative consensus/dissensus) or between teams and domains.

Finally, we note that computer mediation offers arbitrary levels of respondent privacy: investigations could take advantage of respondent-controlled anonymity or identity or amount of identity.

APPLICATION EXTENSIONS

SortDesk technology can be directly extended to the marketing sector of the product development diagram in Figure 3, chapter 1. That is, its use in understanding markets as cultures would represent just another application of anthropology to marketing. Cultural Consensus Theory might be profitably applied to determining relative market-demand expertise among consumers (their competence), gauging homogeneity of market segments (segment mean competence), and to best scale the salience of explicit elements of market demand (use the answer key). Given the rigorously demonstrated power of Cultural Consensus Theory, and the finding that for coherent cultures, as few as six respondents suffice to define cultural truth, opportunity for advances in efficiency in characterizing market demand also seem apparent.

Perhaps the clearest opportunity for extension to marketing lies in attempts to understand computer system and Internet-mediated markets. Consumers in these markets are perforce computer literate, often widely dispersed geographically and consequently hard to sample in person, and the nature of consumer demand in these markets evolves rapidly and remains incompletely explored. As an aid to human-computer interface design, computer usability experts could use SortDesk to understand how consumers think about the software they use or might use.

Finally, groups of people who constitute an Internet-mediated culture could use SortDesk to understand themselves and as an aid to developing their own group's cultural consensus.

MEDIUM ELABORATION OPPORTUNITIES

The most serious shortcoming of SortDesk is its lack of an easy to use card deck generator. Currently, to set up an administration one must have at least passing familiarity with programming in Visual Basic and a copy of that programming language's

development environment. A preferred embodiment of such a generator would allow "on the fly" generation of decks so that a team might brainstorm directly to cards ready for sorting, either simultaneously or by an identified data recorder. Such a generator would provide various options for screen layout of the cards.

Because the look and feel of our software approximates commercially available Windows software, respondents expect the same safety net that they take advantage of when learning by exploration in, or working rapidly with, commercial software. We have had some success with using a variety of documentation to assist respondents but the software has to be made more fault tolerant and must allow respondents to revisit any choice: in the syntax of Windows, various levels of Undo need to be implemented.

It would be well that the software should do real-time analysis so that teams could instantly review the results of their sorting and scaling. Having rapid feedback would allow teams to focus dialog for greater effect.

Although Borgatti advocates expressing domain elements as words, principally to keep the stimuli as abstract and therefore broadly evocative as possible, for cultures having respondents with high mean competence, in which cultural consensus has been reasonably well characterized and found stable, image/video/audio presentation could be investigated to provide more detailed description of the domain. (Borgatti 1992)

A Java version for web-mediated consensus analysis is an obvious next step and would simplify distribution of software and may reduce interoperability problems arising out of respondents' use of different operating systems.

The sorting software would be easier to use if a zoom option were implemented because that would allow informants to better manage the visual complexity of the sort arena. In addition, means should be provided for collapsing piles into an icon (e.g., a folder) with an

optional respondent-provided title. A further extension would be to provide a hierarchy of visually nested icons for construction of taxonomies.

SortDesk needs to store data in a database, providing that doing so does not affect the look and feel of card manipulation.

Frequently enough, respondents wish to put the same card in multiple piles. When this occurs infrequently, it does not present a major problem for Cultural Consensus Theory analysis. Accordingly, the option to clone cards should be implemented.

BIBLIOGRAPHY

Akao, Yoji (1990) *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Portland: Productivity Press.

Allen, Thomas J. (1977) *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization*, Cambridge: MIT Press.

Anonymous (1996a) "Interval Research Corporation Forms Three New Companies; Four Years Since Founding, Lab Foresees Commercial Promise in Distinct Areas of Research and Development," *Business Wire*, November 13 1996, p11130040.

Anonymous (1996b) "Major Design Firm Re-positions for Medical Market; GVO Hires Medical Industry Expert," *Business Wire*, December 2 1996, p12020231.

Arnauld, Eric J. and Sidi Mohammed Iddal (1992) *Niger: Export Marketing of Nigerien Onions*, Washington, DC: U.S.A.I.D., Post-Harvest Institute for Perishables, University of Idaho and Abt Associates.

Arnauld, Eric J. and Melanie Wallendorf (1994), "Market-Oriented Ethnography: Interpretation Building and Market Strategy Formulation," *Journal of Marketing Research* V31n4, pp484-505.

Batchelder, W. H. and A. K. Romney (1986). "The Statistical Analysis of a General Condorcet Model for Dichotomous Choice Situations" in B. Grofman and G. Owen (Eds.), *Information Pooling and Group Decision Making*, Greenwich, Connecticut: JAI Press, Inc., pp. 103-112.

Batchelder, W. H., and A. K. Romney (1988) "Test Theory Without an Answer Key," *Psychometrika* V53: pp71-92.

Batchelder, W. H. and A. K. Romney (1989) "New Results in Test Theory Without an Answer Key," in E. Roskam (Ed.), *Advances in Mathematical Psychology Vol. II. Heidelberg* - New York: Springer-Verlag, pp. 229-248.

Batchelder, W. H., E. Kumbasar, and J. P. Boyd (1997) "Consensus Analysis of Three-Way Social Network Data, *Journal of Mathematical Sociology* V22: pp29-58.

Berlin, Brent, Dennis E. Breedlove, and Peter H. Raven (1969) "Folk Taxonomies and Biological Classification," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp60-66.

Black, Mary B. and Duane Metzger (1969) "Ethnographic Description and the Study of Law," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp137-165.

Blomberg, Jeannette, Lucy Suchman, and Randall H. Trigg (1996) "Reflections on a Work-Oriented Design Project," *Human-Computer Interaction*, V11n3, pp237-265.

Borgatti, S. (1992) *Anthropac 4.0 Methods Guide*, Natick: Analytic Technologies.

Boster, James (1994) "The Successive Pile Sort," *Cultural Anthropology Methods* V6, Issue 2.

Bowker, Geoffrey C., and Susan Leigh Star (downloaded May 8, 1997) "Ethnography of Information Systems," on line course syllabus for LIS 450EI at the University of Illinois Urbana-Champaign. URL: http://alexia.lis.uiuc.edu/~bowker/syllabus.html.

Brewer, D. D., A. K. Romney, and W. H. Batchelder (1991) "Consistency and Consensus: a Replication," *Journal of Quantitative Anthropology*, V3: pp195-205.

Brown, John Seely and Paul Duguid (1994) "Borderline Issues: Social and Material Aspects of Design," *Human-Computer Interaction*, V9n1, pp3-36.

Brown, Shona L., and Kathleen M Eisenhardt (1995) "Product Development: Past Research, Present Findings, and Future Directions," *Academy of Management Review*, V20(n2), pp343-379.

Bucciarelli, Louis L. (1988) "Engineering Design Process," *Making Time: Ethnographies of High-Technology Organizations*, Dubinskas, F. A., ed., Philadelphia, Temple University Press.

Bucciarelli, Louis L. (1994) *Designing Engineers*, Cambridge: MIT Press.

Burawoy, M., "The Anthropology of Industrial Work," *Annual Review of Anthropology*, 1979, pp231-266.

Burton, Michael (1972) "Semantic Dimensions of Occupation Names," *Multidimensional Scaling, Volume II*, Romney, A., R. Shepard, and S. Nerlove, eds., New York: Seminar Press, pp55-71.

Burton, Michael L. and Sara B. Nerlove (1976) "Balanced Designs for Triads Tests: Two Examples from English," *Social Science Research*, V5, pp247-267.

Casson, R. W. (1994) "Cognitive Anthropology," *Handbook of Psychological Anthropology*, Bock, P. K., ed., Westport, CT: Greenwood Press.

Clausing, Don P. (1994) Personal communication, January 1994.

Colby, Benjamin N. (1996) "Cognitive Anthropology," *Encyclopedia of Cultural Anthropology*, Vol. 1, Levinson, D. and M. Ember, eds., sponsored by Human Relations Area Files at Yale University, New York, Henry Holt and Company, pp209-215.

Conklin, Harold C. (1969) "Lexicographical Treatment of Folk Taxonomies," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp41-60.

Donnellon, Anne, (1996) *Team Talk: the Power of Language in Team Dynamics*, Boston:: Harvard Business School Press.

Frake, Charles O. (1969) "The Ethnographic Study of Cognitive Systems," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp28-41.

Frake, Charles O. (1969) "Notes on Queries in Ethnography," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp123-127.

Frake, Charles O. (1969) "Further Discussion of Burling," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp432-433.

Frake, Charles O. (1969) "A Structural Description of Subanun 'Religious Behavior'," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp470-487.

Freeman, Linton (1994) "Collecting Data in Two-Dimensional Euclidean Metric Space," *Cultural Anthropology Methods* V6, Issue 1, pp10-12.

Goodenough, Ward H. (1969) "Yankee Kinship Terminology: A Problem in Componential Analysis," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp255-288.

Goodenough, Ward H. (1969) "Rethinking 'Status" and 'Role': Toward a General Model of the Cultural Organization of Social Relationships," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp311-330.

Gouldner, A. W (1957, 1958) "Cosmopolitans and Locals: Toward an Analysis of Latent Social Roles, I and II," *Administrative Science Quarterly* 2: pp281-306, 444-80.

Green, Paul E., and Frank J. Carmone (1972) "Marketing Research Applications of Nonmetric Scaling Methods," *Multidimensional Scaling, Volume II*, Romney, A., R. Shepard, and S. Nerlove, eds., New York: Seminar Press, pp183-210.

Hauser, John, and Don P. Clausing, "The House of Quality," *Harvard Business Review*, April-May, 1988, pp66-76.

Heath, Rebecca Piirto (1997) "Seeing is Believing," *Marketing Tools Magazine*, March 1997, http://www.demographics.com/publications/MT/97_MT/mt97032.htm.

Hunn, Eugene (1995) Class Notes for Graduate Ethnographic Method Seminar, University of Washington, Fall 1995.

Kay, Paul (1969) "Comments on Colby," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp78-91.

Kidder, John Tracy (1981) *The Soul of a New Machine*, Boston: Atlantic Monthly Press.

Klauer, K. C. and W. H. Batchelder (1996) "Structural Analysis of Subjective Categorical Data, *Psychometrika.* V61: pp199-240.

Kunda, Gideon, (1991) *Engineering Culture*, Philadelphia: Temple University Press.

Lannon, J. (1994) "Mosaics of Meaning: Anthropology and Marketing," *Journal of Brand Management*, V2 n3, pp155-168.

Leong, Siew-Meng (1989) "A Citation Analysis of the Journal of Consumer Research," *Journal of Consumer Research*, V15n4, pp492-502.

Lodge, Milton (1981) *Magnitude Scaling: Quantitative Measurement of Opinions*, Beverly Hills: Sage Publications.

Marples, David L. (1961) "The Decisions of Engineering Design," *IRE Transactions on Engineering Management*, June 1961, pp55-71.

Miller, Marc, John Kaneko, Paul Bartram, Devon D. Brewer, and Joseph Marks, Jr. (1997) "Consensus Analysis of the Pelagic Knowledge of Fishermen and Fishery Scientists in Hawaii," submitted for publication.

Mitchell, Marilyn L. (1998) *Employing Qualitative Methods in the Private Sector*, in press, Sage Press.

Monahan, D. (1997) "Summer 1997 Special Project Report," unpublished Working Paper, Department of Industrial Engineering, Purdue University.

Page, Scott (1995) "Search Strategies for Finding Optima in Fitness Landscapes'" videotaped lecture at the Santa Fe Institute, August 1995, Santa Fe.

Posner, Bruce G. (1996)"Inside a Converted Schwinn Bicycle Factory," *Fast Company*, November 1996, pp105-9.

Rayner, Bruce (1997) "Now Hear This!" *Electronic Business Today*, August 1997, p36.

Romney, A. K. and R. G. D'Andrade (1969) "Cognitive Aspects of English Kin Terms," *Cognitive Anthropology*, Tyler, Stephen A., ed., New York: Holt Rinehart and Winston, pp369-396.

Romney, A., R. Shepard, and S. Nerlove, eds. (1972) *Multidimensional Scaling, Volume II*, New York: Seminar Press

Romney, A. K., S. C. Weller, and W. H. Batchelder (1986) "Culture as Consensus: A Theory of Culture and Informant Accuracy," *American Anthropologist* V88: pp313-338.

Romney, A. K., W. H. Batchelder, and S. C. Weller (1987) "Recent Applications of Consensus Theory," *American Behavioral Scientist* V31: pp163-177.

Romney, A. K. and J. P. Boyd, C. C. Moore, W. H. Batchelder, and T. J. Brazil (1996) "Culture as Shared Cognitive Representation," *Proceedings of the National Academy of Sciences*, V93: pp4699-4705.

Stefflre, Volney J. (1972) "Some Applications of Multidimensional Scaling to Social Science Problems," *Multidimensional Scaling, Volume II*, Romney, A., R. Shepard, and S. Nerlove, eds., New York: Seminar Press, pp211-243.

Suchman, Lucy, A. (1987) *Plans and Situated Actions: the Problem of Human-Machine Communication*, Cambridge: Cambridge University Press.

Sutton, Robert I. and Andrew Hargadon (1996) "Brainstorming Groups in Context: Effectiveness in a Product Design Firm," *Administrative Science Quarterly*, V41n4, pp685-719).

Swan, John E.; McInnis-Bowers, Cecilia; Trawick, I. Fredrick, Jr. (1996) "Ethnography as a Method for Broadening Sales Force Fesearch: Promise and Potential," *Journal of Personal Selling & Sales Management*, V16n2, pp57-65.

Swan, John E.; Bowers, Michael R (1998) "Services Quality and Satisfaction: the Process of People Doing Things Together," *The Journal of Services Marketing*, V12n1 p59-73.

Wasserman, Arnold S. (1994) "In Search of 'The Designer,'" *Human-Computer Interaction*, V9n1, pp120-125.

Weller, S. C (1987) "Shared Knowledge, Intracultural Variation, and Knowledge Aggregation, *American Behavioral Scientist*, 31: pp178-193.

Weller, S. C. and A. K. Romney (1988) *Systematic Data Collection*, Newbury Park, Calif. : Sage Publications.

Weller, S. C., L. M. Pachter, R. T. Trotter, and R. D. Baer (1993). "Empacho in Four Latino Groups: a Study of Intra- and Inter-Cultural Variation in Beliefs," *Medical Anthropology*, V15: pp109-136.

Weller, Susan C., and N. Clay Mann (1997) "Assessing Rater Performance without a 'Gold Standard' Using Consensus Theory," *Medical Decision Making*, V17: pp71-79.

Wilson, Elizabeth J. (1996) "Theory Transitions in Organizational Buying Behavior Research," *Journal of Business & Industrial Marketing*, V11n6, pp7-20.

Zussman, R. (1985) *Mechanics of the Middle Class*, Berkeley: University of California Press.

APPENDIX A: INFORMATION THEORETIC ASPECTS OF PILE SORTS

When two elements share a pile, the information in that pairing is defined as the $\log_2$ transform of the improbability of the two elements randomly being in the same pile. (Burton 1972) The most improbable pairs occur in piles with only two elements and the most likely pairs occur in the pile with all of the elements. (There is no information in "all in one pile" and there is no information in "each its own pile," they are essentially equivalent circumstances.)

For almost all coherent cognitive domains, the largest area of cultural consensus is in agreement about which elements do not belong together. However, concordance notwithstanding, this amounts to little information because "not belongingness" is a feature of the most distributed state of well-ordered domains (e.g., taxonomies).

When doing pile sorting only as far as a single free sort, the data obtained are Bernoulli variables: either two cards share a pile or they do not. The data are typically recorded as domain element by domain element matrices with 1s in cells of elements that share a pile and 0s elsewhere. To account for the additional information in smaller piles, we should increment the 1 in the similarity matrix of pile pairs by the negative of $\log_2(P_{i,\alpha})$ when items j and k are in pile α and decrement all the 0s of similarity matrix by $\log_2(Q_i)$.

Thus, in the similarity matrix **X**, for the j,k cell and the ith respondent

$$X_{j,k,i} = -\log_2(P_{i,\alpha}) + 1 \quad \text{when items } j \text{ and } k \text{ are in pile } \alpha$$
$$= \log_2(Q_i) \quad \text{when items } j \text{ and } k \text{ are in different piles}$$

where:

$$P_{i,\alpha} = \frac{\text{Number of pairs of elements in pile } \alpha \text{ for respondent } i}{\text{Number of pairs of elements in domain}}$$

and

$$Q_i = 1 - \Sigma P_{i,\alpha} \text{ over each respondent's set of pile-pairing probabilities}$$

APPENDIX B: THE AFFINITY DIAGRAM PROCESS

Affinity diagrams became widely used in business in the United States following introduction of the so-called "Seven Management and Planning Tools" (developed by the Japan Union of Scientists and Engineers) by GOAL, the Growth Opportunity Alliance of Greater Lawrence and others. Credit for inspiration and elaboration of the affinity diagram process must go to Jiro Kawakita, an anthropologist among other things.

The affinity diagram is the first of the seven tools and is possibly the easiest for the inexperienced to use, which goes some distance to explaining its popularity. Its purpose is collective elicitation of salient cognitive domain elements for some domain of interest to a group of people, and collective construction of a rough taxonomy of those elements. The benefit in this flows from clarity and consensus about the domain within the group. Here is how to do one:

1. Ask a neutrally phrased question that stimulates thinking about the domain. For example, "What are all the aspects of (this domain)?

2. Have group members brainstorm responses to the question, writing them legibly on cards or PostIts. Ask them to include at least one verb and one noun in each response, to restrict each response to only one idea, and to express responses in fewer than twelve words.

3. Randomly stick the PostIts on a large (4' by 10' or larger) sheet of paper taped to a wall or place cards on a large, open table. Have the group collectively inspect each card, one at a time, checking for adherence to the above criteria and satisfying themselves that they understand what each card means. Avoid argument about the truth, salience, or expression of any card. Remove superfluous cards that exactly duplicate another. Add new cards that this part of the exercise stimulates.

4. Have the group collectively, simultaneously, and silently place cards in clusters that they believe belong together. When a substantial amount of structure has emerged or participants show evidence of wanting badly to talk about the clustering, advance to the next step.

5. Collectively, inspect each cluster for consistency, equivalent level of abstraction of cards within, and evidence of internal taxonomy. Adjust piles based on those inspections, splitting those that should stand apart, joining those that belong together, and nesting those that are parts of others at lower levels of abstraction. Find titles for the piles based on content and taxonomy.

APPENDIX C: DOMAIN STATEMENT LISTS

ASC DERIVATIVE PRODUCT DEVELOPMENT

PROJECT ACTIVITY DOMAIN ELEMENTS

hold the critical design review
hold the preliminary design review
show 30 G peak vibration stability
release production drawing package
brainstorm key performance drivers
do risk analysis of best design approaches
pareto to focus on vital few design features
brainstorm designs as an individual
come to agreement on specifications
find out all the things the customer needs
hold the production readiness review
solve file transfer problems w/customer
alert customer to drastic schedule change
coordinate best design testing w/customer
generate design verification test plan
show that the design has 500 G shock stability
demonstrate thermal cycling stability
write design verification test report
perform the design verification test
test prototype hardware
ship 15 prototype units to customer
deliver 3 proof of concept units as scheduled
customer evaluates prototypes
review design often with those not on project
ask line people for their input
involve automation engineering early on
try to get a marketing person on the team
manufacture initial run units
incorporate and test design pickups
release initial run drawings
generate all production documentation
develop manufacturing processes

develop optimized detail designs
build best guess hardware using DOE methods
improve quartz strength
focus on optimizing the reed design
develop pellet joining methods
optimize can design
review baseline data for 2KI and Mini Q designs
research past design elements
divide work to try alternate design approaches
try Pugh Concept Development
do a FMEA for the HPE accelerometer
generate a baseline design
conceptualize best design approaches
find out what our competition is doing
outline requirement definition

SAE FORMULA CAR '97

IMPORTANCE RANKING DOMAIN ELEMENTS

stay on schedule
learn from past mistakes
work together effectively
learn needed skills ASAP
use reliable design methods
pick energetic leaders
maintain a postive mental attitude
enjoy yourselves
harmonize commitment levels
focus individual initiative on team's work
find welders max tors. rigid. compete sober
document everything
make money
get along
show prospects commitment required
drivers practice their butts off
be car gonzo
plan travel

TECHNICAL GROUP PILE SORT AND TRIADS TEST DOMAIN ELEMENTS

build subassemblies into a car
choose a concept design
distribute our Tech Group's work load
analyze detail designs
generate concept designs
build our Tech Group's team spirit
study the experiences of prior teams
determine performance targets
learn the Competition rules
choose a detail design
identify design constraints
resolve design conflicts with other Groups
plan our Tech Group's activities
create detail designs
evaluate concept designs
produce Competition documents
decide what and how to procure
make progress reports
procure parts and information
decide how to make and assemble parts
develop manufacturing capability
manufacture parts
fabricate subassemblies
test, tune, and debug car with drivers

ADMINISTRATIVE GROUP PILE SORT DOMAIN ELEMENTS

repair clutch cable
change oil, coolant
take engine to cycle sport for intro meeting
schedule work sessions
rebuild the '90 car (engine, etc.)
get the '96 car ready to run
have car/cars ready for driver training
purchase needed parts
write final report
clean both cars many times
inflate and change tires
write checks to reimburse members
give updates on budget situation

find out what groups need to purchase
make a list of vendors
organize purchasing procedures
collect and organize all receipts
review the rules for cost reports
obtain price quotes of everything donated
design the format of our cost report
complete the cost report
organize and run a car wash
organize a raffle
sell raffle tickets at various events
find alternate testing sites
document the arrangements for driving sessions
get risk mgmt., parking & police approval
prepare test plans for driving sessions
make up for uncompleted driving sessions
make signs for rundraising and "pr"
follow up with thank you letters
design and sell t-shirts"
produce a budget for all expenditures
test and tune '97 car with G-Analyst
baseline G-Analyst with '96 car
review safety measures before driving sessions
record all drivers' performances
provide reliable electrical data recording
design & set up an auto-cross driving course

SUMMER 1997 SPECIAL PROJECT

TEAM A DOMAIN ELEMENTS

address the time constraints
deal with cultural differences
plan the web pages project
design the web pages
code the web pages
conduct research
communicate with other companies
communicate within our company
negotiate the overlap with CMSoft
develop 'give and take' among group members

plan the cdrom project
design the cdrom
code the cdrom
relate w/each other in work situations
relate w/each other in social situations
learn java
learn vrml
plan for the vrml model of MGL
design the vrml model of MGL
play with cool software
learn animation software
make a common vision into a reality
find times to meet
cope with our lack of expertise
enjoy the monetary rewards
resolve the conflicts which arise
discover other companies' wants for web
get satisfaction from job completion
balance class and company work
divide tasks among company members
grow consensus within the company
satisfy our customers
figure out how to use the digitizer
deal with lack of computer availability
make everything visually interesting
stay accurate and take artistic license
represent each company equally
leave nothing important out
convey info in a way that all understand

TEAM B DOMAIN ELEMENTS

develop collaboration software
support collaboration software
create scripts that automate routine jobs
do Lotus Notes development
support Lotus Notes
form subteams as needed
develop visualization software
try to understand our mission
design our plan for projects

redesign our plan for projects
develop intelligent SW for IWP
develop software sharing agents
interact with CAD Lab
dream up more collaboration scenarios
resolve overlap with Animedia
work late at night because we like to
look for concrete project activities
choose goals by consensus
make our plans after we choose goals
go to meetings
rearrange the layout of MGL
assemble computer systems
disassemble computer system
cope with loss of visualization people
enjoy finally getting to write programs
collaborate with the other companies
deal with documentation prep boredom
test and debug programs
play with Lotus Notes
forget to do the daily log
try out new software
cope with lack of network SW experience
deal with members' vacations
compete for limited computer access
struggle with budget limitations
compete for meeting space
feel good that research in agents is cool
worry about finishing all our tasks
cope with cultural differences
deal with differences in education level
wonder what collaboration really means
watch exciting demos
find great resources

TEAM C DOMAIN ELEMENTS

determine the nature of our tasks
acquire MW communications equipment
install MW communication equipment
test MW communication equipment

use VTech SW to predict interference
compare VTech SW predictions with actuality
consider licensing VTech software
translate blueprints into AutoCad model
acquire IR communication equipment
install IR communication equipment
test IR communication equipment
learn java
learn VTech software
create wireless LANs
learn software for ad hoc wireless LANs
integrate wireless LANs into furniture
collaborate with IWP
install Breezecom video system
test Breezecom video system
build wireless A/V/data system for shop
test wireless A/V/data system for shops
figure out how to reintegrate late arrivers
understand middle school requirements
build middle school system
collaborate with CPS
collaborate with CMSoft
collaborate with Animedia
test middle school system
cope with the time constraints
deal with time constraints
deal with data collection hassle
manage our fear of failure
cope with possible lack of innovation
understand IWP's requirements
get multi-cell phone transmission equipment
test multi-cell phone transmission equipment
get the technical expertise
ensure that all get to contribute meaningfully
deal w/competition for space and resources
develop remote diagnostic system
get remote diagnostic system parts
build remote diagnostic system
test remote diagnostic system

TEAM D DOMAIN ELEMENTS

determine what software we have
decide what software we need
select a measuring machine or system
select a laptop
select a camera
arrange for capital for purchases
procure needed hardware and software
enjoy the summer's monetary rewards
feel satisfaction in learning new technologies
make something that outlasts this summer
produce something impressive
work w/IWP to develop cool new furniture
collaborate with other companies
form special interest groups
design our web page
get w/CMWave to decide what we're doing
learn Working Model
learn Lotus Notes
learn ProEngineer
learn to use collaboration software
design a widget
design a system to link laptops to shop floor
develop a system linked by cell phones
develop A/V links between shop and design
develop data links between shop and design
drive ProLight with ProE output
develop system for inspection at a distance
develop a strategy/plan
take direction and analyze
decide how to run our meetings
decide how to break the work into chunks
distribute work chunks to individuals
figure out a meeting schedule
decide whether or not we want a hierarchy
take pains to communicate electronically
treat everyone equally
be devoted and consistent
deal with differing personal priorities
value everyone's input
stay open to constructive criticism

cope with insufficient resources
appease the powers that be
manage tight schedules
cope with incomplete info and specs
test our systems under real conditions

TEAM E DOMAIN ELEMENTS

make models of cardboard and straws
make models with CAD packages
experiment with induction power pads
talk a lot with the wireless group
talk a lot with the software agents group
worry that we won't make elegant things
fret about getting to totally wireless
experiment with pivots, swivels and goosenecks
find the relevant electrical codes
design and build a receptacle
make the receptacle safe
design a cool new adaptadesk
design a cool new monodesk
work with Haworth to make desks
consider ergonomics in our designs
put a computer in the monodesk
invent ad hoc networking protocols
give workstations laptop LAN links
make portable workstations
establish the specifications for our designs
take a tour of Haworth
find materials needed for making units
generate a feeling of teamwork
design and make a plug for the flat power grid
learn html
make prototype furniture
develop a project time line
cut sample material
collaborate with other groups for our tasks
collaborate with other groups for their tasks
design a web page for marketing purposes
design layouts for the MGL offices
design and install power grid for MGL

make 3D renderings of offices
work with vendors to design products
work w/vendors to design processes
deal with hidden agendas of IWP members
value differences in experience
downplay differences in credentials
utilize our differences in temperament

SortDesk FAQs

What if I Forgot My Password?

Put in any old password or don't even bother - the password security has been disabled.

The Basis for Deciding What Goes In Piles

Should I decide based on my personal situation or for my whole company?

Always decide based on your company as a whole.

What if we've already accomplished a task?

Always decide from the perspective of the whole set of aspects and tasks - the 'overview' view. Decide based on the whole project set, irrespective of timing.

Positioning the Cards and Piles

How can I get cards to join into piles?

Drag the card you want to add over the pile (or card), highlighting the underlying card(s) in green. Release the mouse button while the underlying card or pile is green and the dragged card will 'snap-join' at the edge nearest the drag card icon.

How can I take a card out of a pile?

Drag the card out of the pile. If the pile has been highlighted green (see the FAQ immediately below), click anywhere on SortDesk other than that pile to 'de-highlight' it before dragging the card you want to remove.

How can I move a pile all at once?

Double-click on any card in the pile and the whole pile will highlight in green. Then just drag any one of the cards in the pile and the whole pile will come along for the ride. Clicking anywhere else on SortDesk other than the highlighted pile will 'de-highlight' the pile. You can also drop a pile on another pile; try it sometime - it's pretty cool.

How can I tell if a group of cards is really a pile?

Sometimes a card which 'connects' two (or more) cards in a pile gets dragged out, leaving a hole in the pile. Don't be fooled - those unconnected cards are still in the same pile. In general, it is a good idea to tidy up moth-eaten piles so that they're visually in one piece. If you're unsure about whether some cards are in a pile, double-click on one of them to see if they all light up green.

What if a card or pile slides partway off the screen?

SortDesk is very literal about putting cards nearest the card edge you drop them on. This means that if you drop a card near an edge where another card is already joined, SortDesk will add the card as indicated and 'bump' the previous card(s). Sometimes this may cause cards to scoot off the screen. At this point, it is best to immediately double-click on the pile and drag all of it on screen. If you don't, you might lose cards off screen where you can't drag them back. Also, if cards hang down off the light gray area of SortDesk onto the dark gray area at the bottom they may obscure the title-entry text box during that phase of the sorting, so it's a good idea to arrange piles so that they're entirely on the actual sorting desk.

Changing Your Mind about "Done"

What if I hit the "Done" button accidentally?

In certain circumstances, "Done" becomes disabled, so a mistaken click makes no difference. "Done" is disabled until you've made at least the minimum number of piles required or have no more than the maximum number allowed. It's also disabled during entry of pile titles and until all the cards have been taken off the deck in "importance" and "difficulty" ranking. At all other times, "Done" is done - there's no going back, so continue with the rest of SortDesk

What if I change my mind about piles after I hit the "Done" button?

See the FAQ immediately above.

Entering Titles for Piles

Why does my title text sometimes disappear after I enter it?

You're probably hitting the "Enter" key on the keyboard instead of clicking on the "Enter Text" button on the screen. The textbox dutifully adds a linefeed/carriage return to the text you entered but, because the box only shows one line at at time, it seems as though your entry has disappeared. In other words, the text you entered hasn't actually disappeared, it's just below the new blank line. You can delete the blank line or not, but click the button on the screen to enter your title.

Understanding the Importance/Difficulty Sections

Must I put cards in the 5 importance or difficulty "slots" that I see on the screen?
No - the lines are just there to help you spread the cards around. One of the advantages of ranking with SortDesk is that you're not limited to the five or seven or nine choices you'd get with a paper-based ranking method. See the FAQ immediately below for more details.

How many levels of importance/difficulty ranking between cards can I show?
The SortDesk screen is divided into almost 10,000 vertical slots - essentially, one per pixel. In theory, if you're careful, you can basically differentiate continuously between cards. Of course, we humans can't really make such fine distinctions in our thinking, but SortDesk would accommodate it if we could.

Do cards have the same (imp., diff.) ranking if they're horizontally in the same pile?
No. Because SortDesk only pays attention to horizontal position, it doesn't matter that the cards are in the same pile. Therefore, because their horizontal positions differ, they have different rankings. On the other hand, a good way to ensure that cards are given the exact same ranking is to stack them vertically in the same pile because the pile-making process forces all the vertical edges into alignment.

The Basis for Deciding Importance or Difficulty

Should I decide based on my personal situation or for my whole company?
Always decide based on your company as a whole

What if we've already accomplished a task?
Always decide from the perspective of the whole set of aspects and tasks - the overview view. Decide based on the relative importance or difficulty for the whole project set.

APPENDIX E: SORTDESK SOURCE CODE

SORTDESK INTRO FRAME SOURCE CODE

```
VERSION 4.00
Begin VB.Form Form1
   Appearance      =   0  'Flat
   BackColor       =   &H00C0C0C0&
   Caption         =   "IntroForm"
   ClientHeight    =   8145
   ClientLeft      =   60
   ClientTop       =   360
   ClientWidth     =   11880
   FillColor       =   &H00808080&
   Height          =   8550
   Left            =   0
   LinkTopic       =   "Form1"
   ScaleHeight     =   8145
   ScaleWidth      =   11880
   Top             =   15
   Width           =   12000
   Begin VB.CommandButton Command2
      Caption         =   "Cancel"
      Height          =   375
      Left            =   6960
      TabIndex        =   6
      Top             =   5040
      Width           =   735
   End
   Begin VB.CommandButton Command1
      Caption         =   "Enter"
      Height          =   375
      Left            =   6120
      TabIndex        =   5
      Top             =   5040
      Width           =   735
   End
   Begin VB.TextBox Text2
      BeginProperty      Font      {0BE35203-8F91-11CE-9DE3-
00AA004BB851}
         Name            =   "MS Sans Serif"
         Size            =   13.5
         Charset         =   0
```

```
         Weight           =   400
         Underline        =   0   'False
         Italic           =   0   'False
         Strikethrough    =   0   'False
      EndProperty
      Height           =   495
      Left             =   3960
      TabIndex         =   2
      Top              =   4080
      Width            =   3735
   End
   Begin VB.TextBox Text1
      BeginProperty      Font      {0BE35203-8F91-11CE-9DE3-
00AA004BB851}
         Name             =   "MS Sans Serif"
         Size             =   13.5
         Charset          =   0
         Weight           =   400
         Underline        =   0   'False
         Italic           =   0   'False
         Strikethrough    =   0   'False
      EndProperty
      Height           =   495
      Left             =   3960
      TabIndex         =   1
      Top              =   3240
      Width            =   3735
   End
   Begin VB.Label Label3
      Alignment        =   1  'Right Justify
      Caption          =   "Your Password"
      BeginProperty      Font      {0BE35203-8F91-11CE-9DE3-
00AA004BB851}
         Name             =   "Arial"
         Size             =   14.25
         Charset          =   0
         Weight           =   400
         Underline        =   0   'False
         Italic           =   0   'False
         Strikethrough    =   0   'False
      EndProperty
      Height           =   375
      Left             =   1440
      TabIndex         =   4
      Top              =   4200
      Width            =   2415
   End
```

```
   Begin VB.Label Label2
      Alignment       =   1  'Right Justify
      Caption         =   "Today's Date"
      BeginProperty       Font       {0BE35203-8F91-11CE-9DE3-
00AA004BB851}
         Name             =   "Arial"
         Size             =   14.25
         Charset          =   0
         Weight           =   400
         Underline        =   0   'False
         Italic           =   0   'False
         Strikethrough    =   0   'False
      EndProperty
      Height          =   375
      Left            =   1920
      TabIndex        =   3
      Top             =   3360
      Width           =   1935
   End
   Begin VB.Label Label1
      Alignment       =   2  'Center
      Caption         =   "Welcome to SortDesk..."
      BeginProperty       Font       {0BE35203-8F91-11CE-9DE3-
00AA004BB851}
         Name             =   "Arial"
         Size             =   48
         Charset          =   0
         Weight           =   400
         Underline        =   0   'False
         Italic           =   0   'False
         Strikethrough    =   0   'False
      EndProperty
      ForeColor       =   &H00800000&
      Height          =   2175
      Left            =   1680
      TabIndex        =   0
      Top             =   480
      Width           =   8535
   End
End
Attribute VB_Name = "Form1"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Command1_Click()
Dim Today As Date
Dim PWord As String
'set date and pw
```

```
    On Error Resume Next
    Today = Form1.Text1.Text
    On Error Resume Next
    PWord = Form1.Text2.Text
'dump date and pw
    If OldPWord = Empty Then Write #1, PWord
    Close #1
    Write #2,
    Write #2, Today
    Write #2, Now
    Write #2,
'put IOBar behind everything
    SortDesk!IOBar.ZOrder 1
    SortDesk.Show
'set focus to the Help button
    SortDesk!Ok.SetFocus
    Unload Form1
'flash the current task label
    For K = 1 To 20
        If          SortDesk!TaskLabel.ForeColor          =
SortDesk!TaskLabel.BackColor Then
            SortDesk!TaskLabel.ForeColor = &HFFFFFF
        Else:         SortDesk!TaskLabel.ForeColor          =
SortDesk!TaskLabel.BackColor
        End If
'pause for ~1/4 second
        PauseTime = 0.25 ' Set duration.
        Start = Timer    ' Set start time.
        Do While Timer < Start + PauseTime
            DoEvents     ' Yield to other processes.
        Loop
    Next K
End Sub

Private Sub Command2_Click()
Write #2, Now
End
End Sub
```

SORTDESK MODULE 1 SOURCE CODE

```
Attribute VB_Name = "MODULE1"
Global IGotIt As Boolean
Global Pile(0 To 60, -14 To 14, -14 To 14) As Integer
Global FSPile(0 To 60, -14 To 14, -14 To 14) As Integer
Global Card2Pile(0 To 60, 0 To 2)  As Integer
```

```
Global FSC2P(0 To 60, 0 To 2)  As Integer
Global PileOrder(0 To 60, 0 To 60) As Variant
Global FSPO(0 To 60, 0 To 60) As Variant
Global PBounds(0 To 60, 0 To 3) As Single
Global Label1 As Control
Global PToggle As Integer
Global OldLeft, OldTop As Single
Dim PIndex, CIndex As Integer
Global Ncards
Global PileMode As Boolean
Global LastX As Integer
Global LastY As Integer
Global HighPile
Global SortStage As Integer
Global QueryPile As Integer
Global SplitIDd As Boolean
Global FSTop(0 To 60) As Integer
Global OldPWord As String
Global FSLeft(0 To 60) As Integer
Global MultiCardPileCount As Integer
Global NPiles As Integer
Global MultiCardPile(1 To 60) As Integer
Global SplitIndex As Integer
Global SecondPileDone As Boolean
Global QueryPile1 As Integer
Global QueryPile2 As Integer
Global LastSplit As Boolean
Global OnPIndex As Integer
Global CurX, CurY As Integer
Global QueryMode As Boolean
Global DistOrder(0 To 60) As Integer
Global DeckX As Integer
Global DeckY As Integer


Sub Add1toPile(OnIndex, DropIndex, X, Y)
'reset ZOrder
    SortDesk!card(DropIndex).ZOrder 0
'dropIndex is the card being dropped, OnIndex is the card
dropped on
    Dim I, PIndex As Integer
'determine Pile being added to
    PIndex = Card2Pile(OnIndex, 0)
'update PileOrder
    Call ExpandPileOrder(PIndex, DropIndex)
'update Card2Pile with new card and pile.
    Card2Pile(DropIndex, 0) = PIndex
```

```
'decide which direction to go
    Select Case Direction(OnIndex, X, Y)
        Case 0  'go up
            NewTop     =     SortDesk!card(OnIndex).Top     -
SortDesk!card(DropIndex).Height
            NewLeft = SortDesk!card(OnIndex).Left
            SortDesk!card(DropIndex).Move NewLeft, NewTop
            Card2Pile(DropIndex, 1) = Card2Pile(OnIndex, 1)
+ 1
            Card2Pile(DropIndex, 2) = Card2Pile(OnIndex, 2)
            PushedIndex = Pile(PIndex, Card2Pile(DropIndex,
1), Card2Pile(DropIndex, 2))
            Pile(PIndex,       Card2Pile(DropIndex,       1),
Card2Pile(DropIndex, 2)) = DropIndex
            If          PBounds(PIndex,          0)          >
SortDesk!card(DropIndex).Top Then
                PBounds(PIndex,             0)             =
SortDesk!card(DropIndex).Top
            End If
            If PushedIndex > -1 Then Call BumpUp(DropIndex,
PushedIndex)
        Case 1  'go right
            NewTop = SortDesk!card(OnIndex).Top
            NewLeft    =    SortDesk!card(OnIndex).Left    +
SortDesk!card(OnIndex).Width
            SortDesk!card(DropIndex).Move NewLeft, NewTop
            Card2Pile(DropIndex, 1) = Card2Pile(OnIndex, 1)
            Card2Pile(DropIndex, 2) = Card2Pile(OnIndex, 2)
+ 1
            PushedIndex = Pile(PIndex, Card2Pile(DropIndex,
1), Card2Pile(DropIndex, 2))
            Pile(PIndex,       Card2Pile(OnIndex,       1),
Card2Pile(DropIndex, 2)) = DropIndex
            If          PBounds(PIndex,          1)          <
SortDesk!card(DropIndex).Left                               +
SortDesk!card(DropIndex).Width Then
                PBounds(PIndex,             1)             =
SortDesk!card(DropIndex).Left                               +
SortDesk!card(DropIndex).Width
            End If
            If    PushedIndex    >    -1    Then    Call
BumpRight(DropIndex, PushedIndex)
        Case 2   'go down
            NewTop     =     SortDesk!card(OnIndex).Top     +
SortDesk!card(OnIndex).Height
            NewLeft = SortDesk!card(OnIndex).Left
            SortDesk!card(DropIndex).Move NewLeft, NewTop
```

```
            Card2Pile(DropIndex, 1) = Card2Pile(OnIndex, 1)
- 1
            Card2Pile(DropIndex, 2) = Card2Pile(OnIndex, 2)
            PushedIndex = Pile(PIndex, Card2Pile(DropIndex,
1), Card2Pile(OnIndex, 2))
            Pile(PIndex,       Card2Pile(DropIndex,       1),
Card2Pile(OnIndex, 2)) = DropIndex
            If          PBounds(PIndex,          2)          <
SortDesk!card(DropIndex).Top                                  +
SortDesk!card(DropIndex).Height Then
                PBounds(PIndex,             2)              =
SortDesk!card(DropIndex).Top
            End If
            If     PushedIndex     >     -1     Then     Call
BumpDown(DropIndex, PushedIndex)
        Case 3   'go left
            NewTop = SortDesk!card(OnIndex).Top
            NewLeft    =    SortDesk!card(OnIndex).Left    -
SortDesk!card(DropIndex).Width
            SortDesk!card(DropIndex).Move NewLeft, NewTop
            Card2Pile(DropIndex, 1) = Card2Pile(OnIndex, 1)
            Card2Pile(DropIndex, 2) = Card2Pile(OnIndex, 2)
- 1
            PushedIndex  =  Pile(PIndex,  Card2Pile(OnIndex,
1), Card2Pile(DropIndex, 2))
            Pile(PIndex,       Card2Pile(OnIndex,       1),
Card2Pile(DropIndex, 2)) = DropIndex
            If          PBounds(PIndex,          3)          <
SortDesk!card(DropIndex).Left                                 +
SortDesk!card(DropIndex).Width Then
                PBounds(PIndex,             3)              =
SortDesk!card(DropIndex).Left                                 +
SortDesk!card(DropIndex).Width
            End If
            If     PushedIndex     >     -1     Then     Call
BumpLeft(DropIndex, PushedIndex)
    End Select
End Sub

Sub BumpDown(BumpIndex, PushedIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(BumpIndex, 0)
'Move PushedCard downward, paste, and alter "row" index in
Card2Pile
    SortDesk!card(PushedIndex).Top                            =
SortDesk!card(BumpIndex).Top                                  +
SortDesk!card(BumpIndex).Height
```

```
    Card2Pile(PushedIndex, 1) = Card2Pile(PushedIndex, 1) -
1
'Retain next Down Index
    NewPushedIndex = Pile(PIndex, Card2Pile(PushedIndex, 1),
Card2Pile(PushedIndex, 2))
'Set cell to new index value
    Pile(PIndex,          Card2Pile(PushedIndex,          1),
Card2Pile(PushedIndex, 2)) = PushedIndex
    If PBounds(PIndex, 2) < SortDesk!card(PushedIndex).Top +
SortDesk!card(PushedIndex).Height Then
        PBounds(PIndex, 2) = SortDesk!card(PushedIndex).Top
+ SortDesk!card(PushedIndex).Height
    End If
'If NewPushedIndex not equal -1, then do it all again
    If NewPushedIndex > -1 Then Call BumpDown(PushedIndex,
NewPushedIndex)
End Sub

Sub BumpLeft(BumpIndex, PushedIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(BumpIndex, 0)
'Move PushedCard leftward, paste, and alter "column" index
in Card2Pile
    SortDesk!card(PushedIndex).Left                        =
SortDesk!card(BumpIndex).Left                              -
SortDesk!card(PushIndex).Width
    Card2Pile(PushedIndex, 2) = Card2Pile(BumpIndex, 2) - 1
'Retain next Left Index
    NewPushedIndex = Pile(PIndex, Card2Pile(PushedIndex, 1),
Card2Pile(PushedIndex, 2))
'Set cell to new index value
    Pile(PIndex,          Card2Pile(PushedIndex,          1),
Card2Pile(PushedIndex, 2)) = PushedIndex
    If PBounds(PIndex, 1) > SortDesk!card(PushedIndex).Left
Then
        PBounds(PIndex, 1) = SortDesk!card(PushedIndex).Left
    End If
'If NewPushedIndex not equal -1, then do it all again
    If NewPushedIndex > -1 Then Call BumpLeft(PushedIndex,
NewPushedIndex)
End Sub

Sub BumpRight(BumpIndex, PushedIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(BumpIndex, 0)
'Move PushedCard rightward, paste, and alter "column" index
in Card2Pile
```

```
    SortDesk!card(PushedIndex).Left                                    =
SortDesk!card(BumpIndex).Left                                          +
SortDesk!card(BumpIndex).Width
    Card2Pile(PushedIndex, 2) = Card2Pile(BumpIndex, 2) + 1
'Retain next right Index
    NewPushedIndex = Pile(PIndex, Card2Pile(PushedIndex, 1),
Card2Pile(PushedIndex, 2))
'Set cell to new index value
    Pile(PIndex,          Card2Pile(PushedIndex,          1),
Card2Pile(PushedIndex, 2)) = PushedIndex
    If PBounds(PIndex, 1) < SortDesk!card(PushedIndex).Left
+ SortDesk!card(PushedIndex).Width Then
        PBounds(PIndex, 1) = SortDesk!card(PushedIndex).Left
+ SortDesk!card(PushedIndex).Width
    End If
'If NewPushedIndex not equal -1, then do it all again
    If NewPushedIndex > -1 Then Call BumpRight(PushedIndex,
NewPushedIndex)
End Sub

Sub BumpUp(BumpIndex, PushedIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(PushedIndex, 0)
'Move PushedCard upward, paste, and alter "row" index in
Card2Pile
    SortDesk!card(PushedIndex).Top                                     =
SortDesk!card(BumpIndex).Top                                           -
SortDesk!card(PushedIndex).Height
    Card2Pile(PushedIndex, 1) = Card2Pile(PushedIndex, 1) +
1
'Retain next Up Index
    NewPushedIndex = Pile(PIndex, Card2Pile(PushedIndex, 1),
Card2Pile(PushedIndex, 2))
'Set cell to new index value
    Pile(PIndex,          Card2Pile(PushedIndex,          1),
Card2Pile(PushedIndex, 2)) = PushedIndex
    If PBounds(PIndex, 0) > SortDesk!card(PushedIndex).Top
Then
        PBounds(PIndex, 0) = SortDesk!card(PushedIndex).Top
    End If
'If NewPushedIndex not equal -1, then do it all again
    If NewPushedIndex > -1 Then Call BumpUp(PushedIndex,
NewPushedIndex)
End Sub

Sub CondensePileOrder(CIndex)
Dim K As Integer
```

```
'identify the Pile needing condensing
    PIndex = Card2Pile(CIndex, 0)
'don't try to condense Pile 0
    If PIndex = 0 Then Exit Sub
'run through the Pile up to first "two consecutive cells = -
1"
    For K = 1 To 50
'find card and pull higher one(s) down one
        Select Case PileOrder(PIndex, K)
'found card to remove; pull next higher index card into this
cell
        Case Is = CIndex
            PileOrder(PIndex, K) = PileOrder(PIndex, K + 1)
'test for CIndex as last card in PileOrder
            If PileOrder(PIndex, K) = -1 Then
                PileOrder(PIndex, 0) = K 'set pointer to the
Kth cell
                Exit Sub 'Done; skedaddle
            End If
'adjacent two identical; continue pulling until they're both
-1 & then exit
        Case Is = PileOrder(PIndex, K - 1)
'forestall possible match between "next cell" index and
CIndex
            If K - 1 > 0 Then
'test for two (-1)s in a row
                If PileOrder(PIndex, K) = -1 Then
                    PileOrder(PIndex, 0) = K - 1 'set
pointer to the K-1th cell
                    Exit Sub 'Done; skedaddle
                End If
            PileOrder(PIndex, K) = PileOrder(PIndex, K + 1)
            End If
        End Select
    Next K
End Sub

Function Direction(OnIndex, X, Y) As Integer
Dim I  As Integer
Static Dist(0 To 3)  As Single
'Compute proportional absolute distances to edges of card
    Dist(0) = Y / SortDesk!card(OnIndex).Height
    Dist(1) = Abs(SortDesk!card(OnIndex).Width - X) /
SortDesk!card(OnIndex).Width
    Dist(2) = Abs(SortDesk!card(OnIndex).Height - Y) /
SortDesk!card(OnIndex).Height
    Dist(3) = X / SortDesk!card(OnIndex).Width
```

```
'Find the smallest distance to edge of card
    SmallestDist = Dist(0)
    Direction = 0
    For I = 1 To 3
        If Dist(I) < SmallestDist Then
            Direction = I
            SmallestDist = Dist(I)
        End If
    Next I
End Function

Sub ExpandPileOrder(PIndex, CIndex)
'Using the index value stored in the zeroth cell, store
CIndex in the next open cell
    PileOrder(PIndex, PileOrder(PIndex, 0)) = CIndex
'Reset the zeroth cell to the new next open cell's index
    PileOrder(PIndex, 0) = PileOrder(PIndex, 0) + 1
End Sub



Sub Main()
Ncards = 45
Dim I, J, K, M, N As Integer
Dim iFirst As Integer
Dim iLast As Integer
Dim iRnd As Integer

'preload Pile array with -1s
    For I = 0 To 60 Step 1
        For J = -14 To 14 Step 1
           For K = -14 To 14 Step 1
                Pile(I, J, K) = -1
           Next K
        Next J
    Next I

'load PileOrder 0 To 60 with -1
    For M = 0 To 60
    'Set PileOrder index cell (M,0) to next available cell
(1)
        PileOrder(M, 0) = 1
    'Fill the rest with -1
        For N = 1 To 60
            PileOrder(M, N) = -1
        Next N
    Next M
```

```
'reset index cell to next open cell for zeroth pile
    PileOrder(0, 0) = Ncards + 1

'initialize random number generator with seed from clock
    Randomize

'randomize the card deck
    Call Unstacker

'set DragIcon to Card type
    For I = 0 To Ncards - 1
        SortDesk!card(I).DragIcon = SortDesk!PIO(1).DragIcon
    Next I

'set PToggle to Lowlight
    PToggle = 1

'set PileMode to False
    PileMode = False
    HighPile = -1
    SplitID = False
    SplitIndex = 0
    SkipRationale = True

'display the intro/password form

'open the pw file for output if first time
    On Error GoTo 10
    Open "a:\d1890" For Input As 1
    Input #1, OldPWord
    If OldPWord = "" Then
10      Close #1
        Open "a:\d1890" For Output As 1
    End If
    Open "a:\cps.sdd" For Append As 2
'set password char
    Form1.Text2.PasswordChar = "*"
'show the form
    Form1.Show
'set focus to the Enter button
    Form1!Command1.SetFocus
    SortStage = 0
End Sub
```

```
Function NewPIndex() As Integer
'Find the least index (above 0) of an empty Pile array
Dim K As Integer
    For K = 1 To 50
        If PileOrder(K, 0) = 1 Then NewPIndex = K: Exit
Function
    Next K
End Function



Sub SortSwap(v1 As Variant, v2 As Variant)
Dim tL, tT As Integer
'store the temp variables
    tL = SortDesk!card(v1).Left
    tT = SortDesk!card(v1).Top
    tI = PileOrder(0, v1 + 1)
'move the cards to the other's TopLeft
    SortDesk!card(v1).Move          SortDesk!card(v2).Left,
SortDesk!card(v2).Top
    SortDesk!card(v2).Move tL, tT
'swap the indices in the zeroth pile
    PileOrder(0, v1 + 1) = PileOrder(0, v2 + 1)
    PileOrder(0, v2 + 1) = tI

End Sub

Sub Stacker()
 Dim tL, tT, tI As Integer
'load PileOrder with right stuff
'keep the zeroth pile occupied
    PileOrder(0, 0) = 2
'empty all the other piles and set their next open cell
indicators to 1
    For I = 1 To 60
        PileOrder(I, 0) = 1
        For J = 1 To 60
            PileOrder(I, J) = -1
        Next J
    Next I
'preload Card2Pile with 0s
    For I = 0 To Ncards - 1
        For J = 0 To 2
            Card2Pile(I, J) = -1
        Next J
    Next I
'preload Pile array with -1s
```

```
    For I = 0 To 60 Step 1
        For J = -14 To 14 Step 1
           For K = -14 To 14 Step 1
                Pile(I, J, K) = -1
           Next K
        Next J
    Next I
'set cards to individual piles
    For N = 1 To Ncards
        PileOrder(N, 1) = N - 1
        PileOrder(N, 0) = 2
        Pile(N, 0, 0) = N - 1
        Card2Pile(N - 1, 0) = N
        Card2Pile(N - 1, 1) = 0
        Card2Pile(N - 1, 2) = 0
    Next N
'stack 'em at the lower center of the SortDesk
    DeckX = Int((SortDesk.Width - SortDesk!card(0).Width) /
2)
    DeckY = SortDesk.Height - SortDesk!card(0).Height - 700
    For K = 1 To Ncards - 1
        SortDesk!card(K).Visible = False
        SortDesk!card(K).Enabled = True
        SortDesk!card(K).BackColor = &HC0FFFF 'card Yellow
        SortDesk!card(K).ForeColor  =  &H800000  'text/pile
Blue
        SortDesk!card(K).BorderStyle = 1
        SortDesk!card(K).BackStyle = 1
        SortDesk!card(K).Move DeckX, DeckY
        SortDesk!card(K).Visible = True
        SortDesk!card(K).ZOrder 1
    Next K
End Sub

Sub StartPile(CIndex)
Dim PIndex  As Integer
'Get least index of available piles
    PIndex = NewPIndex()
'Load that Pile array with this card as anchor member
    Pile(PIndex, 0, 0) = CIndex
'Set Card2Pile to PIndex locus
    Card2Pile(CIndex, 0) = PIndex
'Set Row to 0
    Card2Pile(CIndex, 1) = 0
'Set Column to 0
    Card2Pile(CIndex, 2) = 0
'Set PileOrder(PIndex,0)
```

```
    Call ExpandPileOrder(PIndex, CIndex)
End Sub

Sub Take1FromPile(CIndex)
'send this card to the back
    SortDesk!card(CIndex).ZOrder 1
'remove card from Pile array per (anchor cell, row cell,
column cell)
    Pile(Card2Pile(CIndex,    0),    Card2Pile(CIndex,    1),
Card2Pile(CIndex, 2)) = -1
'remove card from this pile's PileOrder
    CondensePileOrder (CIndex)
'shrink pile
    'ShrinkPile (CIndex)
'restore single card dragicon
    SortDesk!card(CIndex).DragIcon                          =
SortDesk!PIO(1).DragIcon
'drop link between card and pile
    Card2Pile(CIndex, 0) = -1
    Card2Pile(CIndex, 1) = -1
    Card2Pile(CIndex, 2) = -1
End Sub

Sub TogglePile(CIndex)
Dim TPIndex, K As Integer

'What Pile am I toggling?
    TPIndex = Card2Pile(CIndex, 0)

    Select Case PToggle

    Case Is = 0 'Pile highlight as &HFF& 'pile Red or
&H800000 'text/pile Blue
        For K = 1 To PileOrder(TPIndex, 0) - 1
            SortDesk!card(PileOrder(TPIndex, K)).BackColor =
&HC000& 'LightGreen
            SortDesk!card(PileOrder(TPIndex, K)).ForeColor =
&HFFFF& 'text Yellow
            SortDesk!card(PileOrder(TPIndex, K)).BorderStyle
= 1
            SortDesk!card(PileOrder(TPIndex, K)).BackStyle =
1
            SortDesk!card(PileOrder(TPIndex,  K)).Enabled  =
True
        Next K

    Case Is = 1 'Lowlight PostIt Yellow w/ Blue text
```

```
        For K = 1 To PileOrder(TPIndex, 0) - 1
            SortDesk!card(PileOrder(TPIndex, K)).BackColor =
&HC0FFFF 'card Yellow
            SortDesk!card(PileOrder(TPIndex, K)).ForeColor =
&H800000 'text/pile Blue
            SortDesk!card(PileOrder(TPIndex, K)).BorderStyle
= 1
            SortDesk!card(PileOrder(TPIndex, K)).BackStyle =
1
            SortDesk!card(PileOrder(TPIndex,  K)).Enabled  =
True
        Next K

    Case Is = 2 'Flyover highlight as  &HFF& 'pile Red or
&H0000C000& 'LightGreen
                'or &H00FFFFFF& 'desk White'or &H00C0FFC0&
'PaleGreen or &H0080FF80& 'PalerGreen
        For K = 1 To PileOrder(TPIndex, 0) - 1
            SortDesk!card(PileOrder(TPIndex, K)).BackColor =
&HC000& 'LightGreen
            SortDesk!card(PileOrder(TPIndex, K)).ForeColor =
&HFFFF& 'text Yellow
            SortDesk!card(PileOrder(TPIndex, K)).BorderStyle
= 1
            SortDesk!card(PileOrder(TPIndex, K)).BackStyle =
1
            SortDesk!card(PileOrder(TPIndex,  K)).Enabled  =
True
        Next K
    Case Is = 3 'Green for enquiry as &H00008000& 'DarkGreen
        For K = 1 To PileOrder(TPIndex, 0) - 1
            SortDesk!card(PileOrder(TPIndex, K)).BackColor =
&H4000& 'DarkGreen
            SortDesk!card(PileOrder(TPIndex, K)).ForeColor =
&HFFFF& 'text Yellow
            SortDesk!card(PileOrder(TPIndex, K)).BorderStyle
= 1
            SortDesk!card(PileOrder(TPIndex, K)).BackStyle =
1
            SortDesk!card(PileOrder(TPIndex,  K)).Enabled  =
True
        Next K
    Case Is = 4 'low low light; disabled
        For K = 1 To PileOrder(TPIndex, 0) - 1
            SortDesk!card(PileOrder(TPIndex, K)).BackColor =
&HC0C0C0 'grey
```

```
            SortDesk!card(PileOrder(TPIndex, K)).ForeColor =
&HFFFFFF 'desk White
            SortDesk!card(PileOrder(TPIndex, K)).BorderStyle
= 1
            SortDesk!card(PileOrder(TPIndex, K)).BackStyle =
1
            SortDesk!card(PileOrder(TPIndex,  K)).Enabled  =
False
        Next K
    End Select

End Sub

Sub Unstacker()
'load PileOrder (0,N) with zeroth pile info
    For N = 1 To Ncards
        PileOrder(0, N) = N - 1
    Next N
'set pile locations to random order
    iFirst = 1
    iLast = Ncards
    For I = iLast To iFirst + 1 Step -1
'swap random location with last location
        iRnd = Int(iFirst + (Rnd * (iLast - iFirst + 1)))
        SortSwap PileOrder(0, I), PileOrder(0, iRnd)
    Next I
'set cards to individual piles
    For N = 1 To Ncards
        PileOrder(N, 1) = N - 1
        PileOrder(N, 0) = 2
        Pile(N, 0, 0) = N - 1
        Card2Pile(N - 1, 0) = N
        Card2Pile(N - 1, 1) = 0
        Card2Pile(N - 1, 2) = 0
    Next N

'set spacing parameters
    'FirstOffsetX = 150 'first distance to left edge of
screen
    'OffsetX = FirstOffsetX
    'OffsetY = 75 'first distance to top edge of screen
    'HSpace = 75 'horizontal distance between cards
    'VSpace = 75 'vertical distance between cards

'place first row of Pile 0, starting at (0,0), and load
arrays
    'For I = 0 To 8
```

```
    'SortDesk!card(I).Top = OffsetY
    'SortDesk!card(I).Left = OffsetX
    'Pile(0, 1, I) = I 'zeroth pile; row 1, column I
    'Card2Pile(I, 0) = 0 'card index 2 zeroth pile
    'Card2Pile(I, 1) = 0 'row 0
    'Card2Pile(I, 2) = I 'column I
    'OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
    'Next I

'increment downward for next row
    'OffsetY = OffsetY + SortDesk!card(I).Height + VSpace
    'OffsetX = FirstOffsetX

'place second row of Pile 0, starting at (-1,0), and load
arrays
    'For I = 9 To 17
    '    SortDesk!card(I).Top = OffsetY
    '    SortDesk!card(I).Left = OffsetX
    '    Pile(0, -1, I - 8) = I 'zeroth pile; row -1, column
I-8
    '    Card2Pile(I, 0) = 0 'card index 2 zeroth pile
    '    Card2Pile(I, 1) = -1 'row -1
    '    Card2Pile(I, 2) = I - 8 'column I-8
    '    OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
    'Next I

'increment downward for next row
    'OffsetY = OffsetY + SortDesk!card(I).Height + VSpace
    'OffsetX = FirstOffsetX

'place third row of Pile 0, starting at (-2,0), and load
arrays
    'For I = 16 To 23
        'SortDesk!card(I).Top = OffsetY
        'SortDesk!card(I).Left = OffsetX
        'Pile(0, -2, I - 16) = I 'zeroth pile; row -2,
column I-16
        'Card2Pile(I, 0) = 0 'card index 2 zeroth pile
        'Card2Pile(I, 1) = -1 'row -2
        'Card2Pile(I, 2) = I - 16 'column I-16
        'OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
    'Next I

'increment downward for next row
    'OffsetY = OffsetY + SortDesk!card(I).Height + VSpace
    'OffsetX = FirstOffsetX
```

```
'place fourth row of Pile 0, starting at (-3,0), and load
arrays
    'For I = 24 To 31
        'SortDesk!card(I).Top = OffsetY
        'SortDesk!card(I).Left = OffsetX
        'Pile(0, -3, I - 24) = I 'zeroth pile; row -3,
column I-24
        'Card2Pile(I, 0) = 0 'card index 2 zeroth pile
        'Card2Pile(I, 1) = -3 'row -3
        'Card2Pile(I, 2) = I - 24 'column I-24
        'OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
    'Next I

'increment downward for next row
    'OffsetY = OffsetY + SortDesk!card(I).Height + VSpace
    'OffsetX = FirstOffsetX
    'For I = 32 To 39
        'SortDesk!card(I).Top = OffsetY
        'SortDesk!card(I).Left = OffsetX
        'Pile(0, -3, I - 32) = I 'zeroth pile; row -4,
column I-32
        'Card2Pile(I, 0) = 0 'card index 2 zeroth pile
        'Card2Pile(I, 1) = -4 'row -4
        'Card2Pile(I, 2) = I - 32 'column I-32
        'OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
    'Next I

'OffsetY = OffsetY + SortDesk!card(I).Height + VSpace
'OffsetX = 105
'For I = 15 To 17
    'SortDesk!card(I).Top = OffsetY
    'SortDesk!card(I).Left = OffsetX
    'OffsetX = OffsetX + SortDesk!card(I).Width + HSpace
'Next I
PileMode = False
HighPile = -1

End Sub



Public Sub TilePile(OnCardIndex, DropCardIndex)
'Turn off the Highlighting
    PToggle = 2
    'TogglePile (OnCardIndex)
'Which pile am I dropping ON?
```

```
    OnPIndex = Card2Pile(OnCardIndex, 0)
'Can't drop a pile on the zeroth pile
    If OnPIndex = 0 Then Exit Sub
    HighPile = OnPIndex
'Which pile am I dropping?
    DropPIndex = Card2Pile(DropCardIndex, 0)
'Establish origin for tiling; find row & column of card
dropped ON
    InitPRow = Card2Pile(OnCardIndex, 1)
    InitPCol = Card2Pile(OnCardIndex, 2)
    PRow = InitPRow
    PCol = InitPCol
'Force down to be first direction tested
    NextDir = 1
'Maximum number of possible iterations = 14
    For I = 1 To 14
        RunLength = I
'Run twice before changing RunLength
        For J = 1 To 2
'Check/place RunLength number of cells
            For K = 1 To RunLength
'Set the next direction to go
                Select Case NextDir
                Case Is = 0 'move Right next
                PCol = PCol + 1
                Case Is = 1 'move Down next
                PRow = PRow - 1
                Case Is = 2 'move Left next
                PCol = PCol - 1
                Case Is = 3 'move Up next
                PRow = PRow + 1
                End Select
'If the cell is empty, fill it...
                If Pile(OnPIndex, PRow, PCol) = -1 Then
'Put lowest DropCard in next OnPile "hole"
                    Pile(OnPIndex,     PRow,     PCol)     =
PileOrder(DropPIndex, 1)
                    TakelFromPile (PileOrder(DropPIndex, 1))
'Delete DropCard from DropPile
                    NewLeft  =  SortDesk!card(Pile(OnPIndex,
InitPRow,   InitPCol)).Left   +   (PCol   -   InitPCol)   *
SortDesk!card(Pile(OnPIndex, InitPRow, InitPCol)).Width
                    NewTop   =  SortDesk!card(Pile(OnPIndex,
InitPRow,   InitPCol)).Top    -   (PRow   -   InitPRow)   *
SortDesk!card(Pile(OnPIndex, InitPRow, InitPCol)).Height
                    Card2Pile(Pile(OnPIndex, PRow, PCol), 0)
= OnPIndex 'Set Card2Pile for DropCard
```

```
                    Card2Pile(Pile(OnPIndex, PRow, PCol), 1)
= PRow
                    Card2Pile(Pile(OnPIndex, PRow, PCol), 2)
= PCol
                    Call          ExpandPileOrder(OnPIndex,
Pile(OnPIndex, PRow, PCol)) 'Add DropCard index to PileOrder
                    SortDesk!card(Pile(OnPIndex,       PRow,
PCol)).Move NewLeft, NewTop
                    SortDesk!card(Pile(OnPIndex,       PRow,
PCol)).BackColor = &HC000& 'LightGreen &HC0FFFF
                    SortDesk!card(Pile(OnPIndex,       PRow,
PCol)).ForeColor = &HFFFF& 'text Yellow &H800000
                    SortDesk!card(Pile(OnPIndex,       PRow,
PCol)).BorderStyle = 1
                    SortDesk!card(Pile(OnPIndex,       PRow,
PCol)).BackStyle = 1
                    If  PileOrder(DropPIndex,  0)  =  1  Then
Exit Sub 'exit sub when DropPile empty
                End If
            Next K

        Select Case NextDir
        Case  Is  =  0  'move  Down  next;  moved  Right  before
change of direction
        NextDir = 1 'reset pointer to new direction

        Case  Is  =  1  'move  Left  next;  moved  Down  before
change of direction
        NextDir = 2 'reset pointer to new direction

        Case Is = 2 'move Up next; moved Left before change
of direction
        NextDir = 3 'reset pointer to new direction

        Case Is = 3 'move Right next; moved Up before change
of direction
        NextDir = 0 'reset pointer to new direction
        End Select
        Next J
    Next I
'Turn off PileMode
    'PileMode = False
    'HighPile = -1
    End Sub

Public Sub TidyPile(OldPIndex)
'make OnCardIndex = first card of old pile
```

```
    OnCardIndex = PileOrder(OldPIndex, 1)
'take it out of the OldPile
    Take1FromPile (OnCardIndex)
'start me a new pile with the first card as anchor
    StartPile (OnCardIndex)
'make DropCardIndex = next card in old pile
    DropCardIndex = PileOrder(OldPIndex, 1)
'tile the new OnPile with the remnant of old DropPile
    Call TilePile(OnCardIndex, DropCardIndex)
    'PileMode = True
    'HighPile = Card2Pile(OnCardIndex, 0)
End Sub

Public Sub ShrinkPile(CIndex)
'what pile is this?
    PIndex = Card2Pile(CIndex, 0)
'don't tidy the zeroth pile
    If PIndex = 0 Then Exit Sub
'if fewer than two cards in remnant pile then skedaddle
    If PileOrder(PIndex, 0) < 2 Then Exit Sub
'find where to pull from
    Select Case PullDirection(CIndex)
    Case 0  'pull Left
        If      Pile(PIndex,      Card2Pile(CIndex,      1),
Card2Pile(CIndex, 2) + 1) > -1 Then
            Call        PullLeft(CIndex,        Pile(PIndex,
Card2Pile(CIndex, 1), Card2Pile(CIndex, 2) + 1))
            Exit Sub
        End If
    Case 1  'pull Up
        If   Pile(PIndex,   Card2Pile(CIndex,   1)   -   1,
Card2Pile(CIndex, 2)) > -1 Then
            Call         PullUp(CIndex,         Pile(PIndex,
Card2Pile(CIndex, 1) - 1, Card2Pile(CIndex, 2)))
            Exit Sub
        End If
    Case 2   'pull Right
        If      Pile(PIndex,      Card2Pile(CIndex,      1),
Card2Pile(CIndex, 2) - 1) > -1 Then
            Call        PullRight(CIndex,        Pile(PIndex,
Card2Pile(CIndex, 1), Card2Pile(CIndex, 2) - 1))
            Exit Sub
        End If
    Case 3   'pull Down
        If   Pile(PIndex,   Card2Pile(CIndex,   1)   +   1,
Card2Pile(CIndex, 2)) > -1 Then
```

```
            Call        PullDown(CIndex,        Pile(PIndex,
Card2Pile(CIndex, 1) + 1, Card2Pile(CIndex, 2)))
            Exit Sub
        End If


    End Select
End Sub
'step through the Pile array, looking for shrinks and taking
them
    'For J = 14 To -14 Step -1
        'If J > 0 Then
            'OffsetY = -1
            'Else: OffsetY = 1
        'End If
        'For K = -14 To 14 Step 1
            'If K < 0 Then
                'OffsetX = 1
            'Else:  OffsetX = -1
            'End If
            'If Pile(PIndex, J, K) <> -1 Then
'if the diagonal is open, move there, empty old cell
                'If Pile(PIndex, J + OffsetY, K + OffsetX) =
-1 Then
                    'Pile(PIndex, J + OffsetY, K + OffsetX)
= Pile(PIndex, J, K)
                    'Card2Pile(Pile(PIndex, J, K), 1) = J +
OffsetY
                    'Card2Pile(Pile(PIndex, J, K), 2) = K +
OffsetX
                    'NewLeft = SortDesk!card(Pile(PIndex, J,
K)).Left + OffsetX * SortDesk!card(Pile(PIndex, J, K)).Width
                    'NewTop = SortDesk!card(Pile(PIndex, J,
K)).Top - OffsetY * SortDesk!card(Pile(PIndex, J, K)).Height
                    'SortDesk!card(Pile(PIndex, J, K)).Move
NewLeft, NewTop
                    'Pile(PIndex, J, K) = -1
'if the adjacent column is open, move there, empty old cell
                'Else:
                    'If Pile(PIndex, J, K + OffsetX) = -1
Then
                    'Pile(PIndex,   J,   K   +   OffsetX)   =
Pile(PIndex, J, K)
                    'Card2Pile(Pile(PIndex, J, K), 2) = K +
OffsetX
                    'NewLeft = SortDesk!card(Pile(PIndex, J,
K)).Left + OffsetX * SortDesk!card(Pile(PIndex, J, K)).Width
```

```
                    'NewTop = SortDesk!card(Pile(PIndex, J,
K)).Top
                    'SortDesk!card(Pile(PIndex, J, K)).Move
NewLeft, NewTop
                    'Pile(PIndex, J, K) = -1
'if the adjacent row is open, move there, empty old cell
                    'Else:
                        'If Pile(PIndex, J + OffsetY, K) = -
1 Then
                            'Pile(PIndex, J + OffsetY, K) =
Pile(PIndex, J, K)
                            'Card2Pile(Pile(PIndex, J, K),
1) = J + OffsetY
                            'NewLeft                    =
SortDesk!card(Pile(PIndex, J, K)).Left
                            'NewTop                     =
SortDesk!card(Pile(PIndex, J, K)).Top - OffsetY *
SortDesk!card(Pile(PIndex, J, K)).Height
                            'SortDesk!card(Pile(PIndex, J,
K)).Move NewLeft, NewTop
                            'Pile(PIndex, J, K) = -1
                        'End If
                    'End If
                'End If
            'End If
        'Next J
    'Next K

Public Function PullDirection(CIndex)
Dim I  As Integer
Dim FoundRight As Boolean
Dim FoundDown As Boolean
Dim FoundLeft As Boolean
Dim FoundUp As Boolean
Static EdgeDist(0 To 3)  As Single
'what pile is this?
    PIndex = Card2Pile(CIndex, 0)
    FoundRight = False
    FoundDown = False
    FoundLeft = False
    FoundUp = False
'compute absolute distances to edges of pile
    For K = 1 To 28
'check rightward
        If FoundRight = False Then
            If    Pile(PIndex,    Card2Pile(CIndex,    1),
Card2Pile(CIndex, 2) + K) = -1 Then
```

```
                EdgeDist(0) = K
                FoundRight = True
            End If
        End If
'check downward
        If FoundDown = False Then
            If  Pile(PIndex,  Card2Pile(CIndex,  1)  +  K,
Card2Pile(CIndex, 2)) = -1 Then
                EdgeDist(1) = K
                FoundDown = True
            End If
        End If
'check leftward
        If FoundLeft = False Then
            If    Pile(PIndex,    Card2Pile(CIndex,    1),
Card2Pile(CIndex, 2)) - K = -1 Then
                EdgeDist(2) = K
                FoundDown = True
            End If
        End If
'check upward
        If FoundUp = False Then
            If  Pile(PIndex,  Card2Pile(CIndex,  1)  -  K,
Card2Pile(CIndex, 2)) = -1 Then
                EdgeDist(3) = K
                FoundUp = True
            End If
        End If
'exit out of loop if got the distances
        If  FoundRight  =  True  And  FoundDown  =  True  And
FoundLeft = True And FoundUp = True Then
        K = 28
        End If
    Next K
'find the least distance to edge of pile
    nEdges = 0
    For I = 0 To 3
        If EdgeDist(I) = 1 Then nEdges = nEdges + 1
    Next I
'pick a final direction
'this one's a peninsula; no change to pile display
    'If nEdges > 1 Then PullDirection = 4
        'Exit Function
    'End If
'Find the greatest distance to an edge of the pile
    GreatestDist = EdgeDist(0)
    PullDirection = 0
```

```
    For I = 1 To 3
        If EdgeDist(I) > GreatestDist Then
            If EdgeDist(I) > 1 Then
                GreatestDist = EdgeDist(I)
                PullDirection = I
            End If
        End If
    Next I
End Function

Public Sub PullUp(SuckIndex, PulledIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(PulledIndex, 0)
'Move PulledCard upward
    SortDesk!card(PulledIndex).Top                                    =
SortDesk!card(SuckIndex).Top
'Retain next Index of next card down, if any
    NewPulledIndex = Pile(PIndex, Card2Pile(PulledIndex, 1)
- 1, Card2Pile(PulledIndex, 2))
'Increment row index
    Card2Pile(PulledIndex, 1) = Card2Pile(PulledIndex, 1) +
1
'Set Pile cell pointed to by the pulled index to its new
resident (PulledIndex)
    Pile(PIndex,          Card2Pile(PulledIndex,          1),
Card2Pile(PulledIndex, 2)) = PulledIndex
    If NewPulledIndex > -1 Then Call PullUp(PulledIndex,
NewPulledIndex)
End Sub

Public Sub PullLeft(SuckIndex, PulledIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(PulledIndex, 0)
'move PulledCard upward
    SortDesk!card(PulledIndex).Left                                   =
SortDesk!card(SuckIndex).Left
'retain next Index of next card down, if any
    NewPulledIndex = Pile(PIndex, Card2Pile(PulledIndex, 1),
Card2Pile(PulledIndex, 2) + 1)
'decrement column index
    Card2Pile(PulledIndex, 1) = Card2Pile(PulledIndex, 1) -
1
'set Pile cell pointed to by the pulled index to its new
resident (PulledIndex)
    Pile(PIndex,          Card2Pile(PulledIndex,          1),
Card2Pile(PulledIndex, 2)) = PulledIndex
```

```
    If NewPulledIndex > -1 Then Call PullLeft(PulledIndex,
NewPulledIndex)
End Sub

Public Sub PullDown(SuckIndex, PulledIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(PulledIndex, 0)
'move PulledCard Downward
    SortDesk!card(PulledIndex).Top                          =
SortDesk!card(SuckIndex).Top
'retain next Index of next card down, if any
    NewPulledIndex = Pile(PIndex, Card2Pile(PulledIndex, 1)
+ 1, Card2Pile(PulledIndex, 2))
'decrement row index
    Card2Pile(PulledIndex, 1) = Card2Pile(PulledIndex, 1) -
1
'set Pile cell pointed to by the pulled index to its new
resident (PulledIndex)
    Pile(PIndex,          Card2Pile(PulledIndex,          1),
Card2Pile(PulledIndex, 2)) = PulledIndex
    If NewPulledIndex > -1 Then Call PullDown(PulledIndex,
NewPulledIndex)
End Sub

Public Sub PullRight(SuckIndex, PulledIndex)
Dim PIndex As Integer
    PIndex = Card2Pile(PulledIndex, 0)
'move PulledCard upward
    SortDesk!card(PulledIndex).Left                         =
SortDesk!card(SuckIndex).Left
'retain next Index of next card to the Left, if any
    NewPulledIndex = Pile(PIndex, Card2Pile(PulledIndex, 1),
Card2Pile(PulledIndex, 2) - 1)
'increment column index
    Card2Pile(PulledIndex, 2) = Card2Pile(PulledIndex, 2) +
1
'set Pile cell pointed to by the pulled index to its new
resident (PulledIndex)
    Pile(PIndex,          Card2Pile(PulledIndex,          1),
Card2Pile(PulledIndex, 2)) = PulledIndex
    If NewPulledIndex > -1 Then Call PullRight(PulledIndex,
NewPulledIndex)
End Sub

Public Sub GetRationale()
'What stage of pile sorting?
Select Case SortStage
```

```
    Case Is = 0 'free sort end

        For K = 1 To 60
'turn next pile rationale Green
            If PileOrder(K, 0) > 1 Then
                PToggle = 3
                TogglePile (PileOrder(K, 1))
'pop text box & stage 0 label
                SortDesk!Query(0).Left                          =
SortDesk!Rationale.Left
                SortDesk!Query(0).Top                           =
SortDesk!Rationale.Top - SortDesk!Query(0).Height
                SortDesk!Rationale.Visible = True
                SortDesk!Query(0).Visible = True
                SortDesk!TextDone.Visible = True
            'on event "end of input," continue
'reset pile color to PostIt Yellow
                'PToggle = 1
                'TogglePile (Pile(K, 0, 0))
            End If
        Next K
        SortStage = 1

    Case Is = 1 'after each successive split

'find start pile and new pile indices

        'turn both piles green Green
        'pop text box and stage 1 label
        'on event "end of input," continue

    Case Is = 2 'after each successive combining

        'find combo pile index
        'turn that pile rationale Green
        'pop text box & stage 2 label

End Select
End Sub

Public Function WritePile(M)
'Write the pile data
    Select Case PileOrder(M, 0)
        Case Is = 2
        Write #2, PileOrder(M, 1)
        Case Is = 3
        Write #2, PileOrder(M, 1), PileOrder(M, 2)
```

```
        Case Is = 4
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M, 3)
        Case Is = 5
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M, 3), PileOrder(M, 4)
        Case Is = 6
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M, 3), PileOrder(M, 4), PileOrder(M, 5)
        Case Is = 7
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M, 6)
        Case Is = 8
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M, 6), PileOrder(M, 7)
        Case Is = 9
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M, 6), PileOrder(M, 7), PileOrder(M, 8)
        Case Is = 10
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M, 9)
        Case Is = 11
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M, 9), PileOrder(M, 10)
        Case Is = 12
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M, 9), PileOrder(M, 10), PileOrder(M, 11)
        Case Is = 13
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,   10),   PileOrder(M,   11),
PileOrder(M, 12)
        Case Is = 14
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
```

```
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M, 12), PileOrder(M, 13)
        Case Is = 15
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M, 12), PileOrder(M, 13), PileOrder(M, 14)
        Case Is = 16
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M,  12),  PileOrder(M,  13),  PileOrder(M,  14),
PileOrder(M, 15)
        Case Is = 17
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M,  12),  PileOrder(M,  13),  PileOrder(M,  14),
PileOrder(M, 15), PileOrder(M, 16)
        Case Is = 18
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M,  12),  PileOrder(M,  13),  PileOrder(M,  14),
PileOrder(M, 15), PileOrder(M, 16), PileOrder(M, 17)
        Case Is = 19
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M,  12),  PileOrder(M,  13),  PileOrder(M,  14),
PileOrder(M,  15),  PileOrder(M,  16),  PileOrder(M,  17),
PileOrder(M, 18)
        Case Is = 20
        Write  #2,  PileOrder(M,  1),  PileOrder(M,  2),
PileOrder(M,  3),  PileOrder(M,  4),  PileOrder(M,  5),
PileOrder(M,  6),  PileOrder(M,  7),  PileOrder(M,  8),
PileOrder(M,  9),  PileOrder(M,  10),  PileOrder(M,  11),
PileOrder(M,  12),  PileOrder(M,  13),  PileOrder(M,  14),
PileOrder(M,  15),  PileOrder(M,  16),  PileOrder(M,  17),
PileOrder(M, 18), PileOrder(M, 19)
        Case Is = 21
```

```
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,  10),   PileOrder(M,  11),
PileOrder(M,  12),   PileOrder(M,  13),   PileOrder(M,  14),
PileOrder(M,  15),   PileOrder(M,  16),   PileOrder(M,  17),
PileOrder(M, 18), PileOrder(M, 19), PileOrder(M, 20)
        Case Is = 22
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,  10),   PileOrder(M,  11),
PileOrder(M,  12),   PileOrder(M,  13),   PileOrder(M,  14),
PileOrder(M,  15),   PileOrder(M,  16),   PileOrder(M,  17),
PileOrder(M,  18),   PileOrder(M,  19),   PileOrder(M,  20),
PileOrder(M, 21)
        Case Is = 23
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,  10),   PileOrder(M,  11),
PileOrder(M,  12),   PileOrder(M,  13),   PileOrder(M,  14),
PileOrder(M,  15),   PileOrder(M,  16),   PileOrder(M,  17),
PileOrder(M,  18),   PileOrder(M,  19),   PileOrder(M,  20),
PileOrder(M, 21), PileOrder(M, 22)
        Case Is = 24
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,  10),   PileOrder(M,  11),
PileOrder(M,  12),   PileOrder(M,  13),   PileOrder(M,  14),
PileOrder(M,  15),   PileOrder(M,  16),   PileOrder(M,  17),
PileOrder(M,  18),   PileOrder(M,  19),   PileOrder(M,  20),
PileOrder(M, 21), PileOrder(M, 22), PileOrder(M, 23)
        Case Is = 25
        Write   #2,   PileOrder(M,   1),   PileOrder(M,   2),
PileOrder(M,   3),   PileOrder(M,   4),   PileOrder(M,   5),
PileOrder(M,   6),   PileOrder(M,   7),   PileOrder(M,   8),
PileOrder(M,   9),   PileOrder(M,  10),   PileOrder(M,  11),
PileOrder(M,  12),   PileOrder(M,  13),   PileOrder(M,  14),
PileOrder(M,  15),   PileOrder(M,  16),   PileOrder(M,  17),
PileOrder(M,  18),   PileOrder(M,  19),   PileOrder(M,  20),
PileOrder(M,  21),   PileOrder(M,  22),   PileOrder(M,  23),
PileOrder(M, 24)
   End Select
End Function
```

```
Public Sub TriadsTest()
SortStage = 4

For K = 0 To Ncards - 1
    SortDesk!card(K).Visible = False
    SortDesk!card(K).Enabled = False
Next K
'here is the call to BIBD or file
Call Unstacker
SortDesk!card(PileOrder(0, 1)).Move 1200, 2280
SortDesk!card(PileOrder(0, 2)).Move 4680, 2280
SortDesk!card(PileOrder(0, 3)).Move 8160, 2280
For K = 1 To 3
    SortDesk!card(PileOrder(0, K)).Width = 2520
    SortDesk!card(PileOrder(0, K)).Height = 1365
    SortDesk!card(PileOrder(0, K)).Font.Size = 19
    SortDesk!card(PileOrder(0, K)).Enabled = True
    SortDesk!card(PileOrder(0,   K)).BackColor   =   &HC0FFFF
'card Yellow
    SortDesk!card(PileOrder(0,   K)).ForeColor   =   &H800000
'text/pile Blue
    SortDesk!card(PileOrder(0, K)).BorderStyle = 1
    SortDesk!card(PileOrder(0, K)).BackStyle = 1
    SortDesk!card(PileOrder(0, K)).Visible = True
Next K
QueryMode = False

End Sub

Public Sub Importance()
'it's stage 5, the importance weighting/ranking
    SortStage = 5
'make a deck at the upper left
    Call Stacker
'no queries here...
    QueryMode = False
'turn on the labels which give verbal cues
'turn on the labels which cue graphically
'display TaskText
'let 'em drag 'em all over the desk
End Sub
```

SORTDESK FRAME SOURCE CODE

```
Private Sub card_Click(Index As Integer)
'write the tetrad if a triads test
    If SortStage = 4 Then
'change the TaskText
        Write   #2,   PileOrder(0,   1),   PileOrder(0,   2),
PileOrder(0, 3), Index
'forever refresh the triads
        Call TriadsTest
        Exit Sub
    End If
'if this is a caption capture, do it and exit
    If QueryMode = True Then
        SortDesk!Rationale.SelText                                    =
SortDesk!card(Index).Tag
        Exit Sub
    End If

Exit Sub

End Sub

Private Sub card_DblClick(Index As Integer)
'if this is a caption capture, do it and exit
    If QueryMode = True Then
        Exit Sub
    End If
'which pile or card am I double clicking on?
    PIndex = Card2Pile(Index, 0)
'if it's just a card, don't allow PileMode
    If PileOrder(PIndex, 0) < 3 Then Exit Sub
'no action if not THE Pile while in PileMode
    If PIndex <> HighPile And PileMode = True Then Exit Sub
'disable all the other piles if in splitting stage
    If SortStage = 1 Then
        For K = 1 To 49
            If  PileOrder(K,  0)  >  1  And  K  <>  PIndex  Then
'pile is not empty and not click pile
                For  I  =  1  To  PileOrder(K,  0)  -  1  'so
skeletonize it & disable its cards
                    SortDesk!card(PileOrder(K, I)).BackColor
= &HC0C0C0
                    SortDesk!card(PileOrder(K, I)).ForeColor
= &HFFFFFF 'desk White
                    SortDesk!card(PileOrder(K, I)).Enabled =
False
```

```
                Next I
            End If
        Next K
'set SplitIDd = True
        SplitIDd = True
        Exit Sub
     End If
'toggle PileMode
    If PileMode = True Then
        PileMode = False
    Else: PileMode = True
    End If
'and set HighPile
    If PileMode = False Then
        HighPile = -1
    Else: HighPile = Card2Pile(Index, 0)
    End If
'paint and enable as appropriate
    Select Case PileMode
    Case Is = True
        PToggle = 0
        TogglePile (Index)
    Case Is = False
        PToggle = 1
        TogglePile (Index)
    End Select
'Track pile changes with temporary windows
    'SortDesk!PIO(0).Caption = PIndex
    'SortDesk!PIO(1).Caption = PileOrder(PIndex, 0)
    'SortDesk!PIO(2).Caption = PileOrder(PIndex, 1)
    'SortDesk!PIO(3).Caption = PileOrder(PIndex, 2)
    'SortDesk!PIO(4).Caption = PileOrder(PIndex, 3)
    'SortDesk!PIO(6).Caption = PileOrder(PIndex, 4)
End Sub

Private Sub card_DragDrop(Index As Integer, Source As
Control, X As Single, Y As Single)
'what OnPile/card am I over?
    OnPIndex = Card2Pile(Index, 0)
'what DropPile/card is flying?
    DropPIndex = Card2Pile(Source.Index, 0)
'trap and fix erroneous moves
'can't drop card on itself or zeroth pile; no drop in pile
mode if not from HighPile,
    If Index = Source.Index Or (OnPIndex = 0 Or (PileMode =
True And (HighPile <> DropPIndex Or OnPIndex = DropPIndex)))
Then
```

```
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if dropping on self, stay highlighted
            If OnPIndex = DropPIndex Then Exit Sub
            PToggle = 1
            TogglePile (Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select
        Exit Sub 'Finished recovering from bogus drops;
skedaddle
    End If

'the rest of this takes care of valid drops:
    Select Case PileMode
'it is a pile to pile add so "tile the pile"
    Case Is = True
        Call TilePile(Index, Source.Index)
        'HighPile = -1
        'PileMode = False
'just one card to add
    Case Is = False
'take the DropCard from the pile it just left
        TakelFromPile (Source.Index)
'add it to the OnPile
        Call AddltoPile(Index, Source.Index, X, Y)
'repaint the new pile as lowlighted cards
        PToggle = 1
        TogglePile (Source.Index)
    End Select
'if not dropping a card on its own pile then tidy it if
required
    'If OnPIndex <> DropPIndex Then ShrinkPile (DropPIndex)
'Track pile changes with temporary windows
    'SortDesk!PIO(0).Caption = Card2Pile(Source.Index, 0)
    'SortDesk!PIO(1).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 0)
    'SortDesk!PIO(2).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 1)
```

```
    'SortDesk!PIO(3).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 2)
    'SortDesk!PIO(4).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 3)
    'SortDesk!PIO(6).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 4)
End Sub

Private Sub card_DragOver(Index As Integer, Source As
Control, X As Single, Y As Single, State As Integer)
'determine Pile underneath
    OnPIndex = Card2Pile(Index, 0)
'no Enter/Leave over zeroth pile
    'If OnPIndex = 0 And Index <> Source.Index Then
'set DragIcon to "illegal drag area icon"
        'SortDesk!card(Source.Index).DragIcon                   =
SortDesk!PIO(2).DragIcon
        'Exit Sub
    'End If
'determine flying Pile
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'if entering over same pile in pilemode do nothing
    If PileMode = True And OnPIndex = DropPIndex Then Exit
Sub
'if entering over same card do nothing
    If Source.Index = Index Then Exit Sub
'if entering as a pile or card over other piles or cards,
highlight underliers red
    PToggle = 2
    TogglePile (Index)
    SortDesk!card(Source.Index).BackColor = &HC0C0C0
    SortDesk!card(Source.Index).ForeColor = &HFFFFFF 'desk
White

    Case LEAVE
'if leaving own pile, iconize leaver(s); otherwise lowlight
underlier
'if leaving own pile as a pile, then "box" the HighPile
    If HighPile = OnPIndex Then
```

```
        For K = 1 To PileOrder(OnPIndex, 0) - 1
            SortDesk!card(PileOrder(OnPIndex,  K)).BackColor
= &HC0C0C0
            SortDesk!card(PileOrder(OnPIndex,  K)).ForeColor
= &HFFFFFF 'desk White
        Next K
    Else
'if just moving a card, lowlight the rest of the pile and
"box" the card
        If OnPIndex <> 0 Then
            PToggle = 1
            TogglePile (Index)
        End If
        If OnPIndex = DropPIndex Then
            SortDesk!card(Source.Index).BackColor = &HC0C0C0
            SortDesk!card(Source.Index).ForeColor = &HFFFFFF
'desk White
        End If
    End If

    Case OVER
        Select Case PileMode
        Case Is = True 'drag as a pile
            SortDesk!card(Index).DragIcon                    =
SortDesk!PIO(0).DragIcon
        Case Is = False 'drag as a card
            SortDesk!card(Index).DragIcon                    =
SortDesk!PIO(1).DragIcon
        End Select
    End Select
End Sub

Private Sub card_MouseDown(Index As Integer, Button As
Integer, Shift As Integer, X As Single, Y As Single)
'forestall any card or pile movement in QueryMode
    If QueryMode = True Then Exit Sub
'forestall any card or pile movement until split pile has
been id'd if in Stage 1
    If SortStage = 1 And SplitIDd = False Then Exit Sub
'forestall any moves in SortStage 4
    If SortStage = 4 Then Exit Sub
'what Pile/card am I on?
    PIndex = Card2Pile(Index, 0)
'Reset Highpile if over other pile when HighPile <> PIndex
AND PileMode = True
    If HighPile <> PIndex And PileMode = True Then
        PToggle = 1
```

```
        TogglePile (PileOrder(HighPile, 1))
        PileMode = False
        HighPile = -1
    End If
'if over pile while in SortStage 2 (combining), set to
HighPile
    If PileOrder(PIndex, 0) > 2 And SortStage = 2 Then
        PToggle = 0
        TogglePile (Index)
        PileMode = True
        HighPile = PIndex
    End If
'get current mouse x & y
    OldX = CurX
    OldY = CurY
'pause for ~1/10 second
'    PauseTime = 0.05  ' Set duration.
'    Start = Timer   ' Set start time.
'    Do While Timer < Start + PauseTime
'        DoEvents    ' Yield to other processes.
'    Loop
'test for movement and bug out if very little
'    If Abs(OldX - CurX) < 1 And Abs(OldY - CurY) < 1 Then
Exit Sub

'ok to react; set the DragIcon
    Select Case PileMode
    Case Is = True 'drag as a pile
        SortDesk!card(Index).DragIcon                    =
SortDesk!PIO(0).DragIcon
    Case Is = False 'drag as a card
        SortDesk!card(Index).DragIcon                    =
SortDesk!PIO(1).DragIcon
    End Select
'drag the object
    SortDesk!card(Index).Drag 1
    DragX = X
    DragY = Y
    OldTop = SortDesk!card(Index).Top
    OldLeft = SortDesk!card(Index).Left
End Sub


Private Sub card_MouseMove(Index As Integer, Button As
Integer, Shift As Integer, X As Single, Y As Single)
    CurX = X
    CurY = Y
```

```
End Sub


Private Sub card_MouseUp(Index As Integer, Button As
Integer, Shift As Integer, X As Single, Y As Single)
SortDesk!card(Index).Drag 2
'Call card_Click(Index)
End Sub

Private Sub Command1_Click()

End Sub

Private Sub Done_Click()
'no action if QueryMode = True
    If QueryMode = True Then Exit Sub
    If HighPile <> -1 Then
'reset the HighPile to PostIt
        PToggle = 1
        TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
        HighPile = -1
'reset PileMode to false
        PileMode = False
    End If

    Select Case SortStage

    Case Is = 0 'first click; end of free sort
'how many piles are there; exit if illegal free sort
    Call PileCount
    If NPiles > Ncards - 1 Or NPiles < 2 Then Exit Sub
    PToggle = 4
    For K = 0 To Ncards - 1
        Call TogglePile(K)
    Next K
'enable pile caption capture mode
    QueryMode = True
'query hightlight the first pile
    QueryPile = NextQPile()
    PToggle = 1
    TogglePile (PileOrder(QueryPile, 1))
'Set TaskText depending upon single or multi-card pile
    If PileOrder(QueryPile, 0) < 3 Then
        SortDesk!TaskText.Caption = "In the text box below,
please enter a title for the highlighted single-card pile.
Don't worry about speling or tpyos, just blaze away.  QUICK
```

```
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Title"" button at the
right of the text box."
'show Query(2)
        Query(2).Visible = True
    Else:   SortDesk!TaskText.Caption = "In the text box
below, please enter a title for the highlighted multi-card
pile.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
        Query(2).Visible = True
    End If
'show Rationale
    Rationale.Visible = True
'give cursor focus in Rationale
    Rationale.SetFocus
'show the Enter button
    TextDone.Visible = True
'set QueryMode to True
    QueryMode = True
'show help
    Label1(2).Visible = True
    Label1(2).ZOrder
    Label1(3).Visible = True
    Label1(3).ZOrder
    TaskText.Visible = True
    TaskText.ZOrder
    TaskLabel.Visible = True
    TaskLabel.ZOrder
    SortDesk!Ok.Visible = True
    SortDesk!Ok.ZOrder
'     Label1(2).Visible = True
'     Label1(3).Visible = True
'     TaskText.Visible = True
'     TaskLabel.Visible = True
'     SortDesk!Ok.Visible = True
'set IGotIt to False
    IGotIt = False
'flash the current task label
    For K = 1 To 20
        If          SortDesk!TaskLabel.ForeColor          =
SortDesk!TaskLabel.BackColor Then
            SortDesk!TaskLabel.ForeColor = &HFFFFFF
        Else:         SortDesk!TaskLabel.ForeColor          =
SortDesk!TaskLabel.BackColor
```

```
        End If
'pause for ~1/4 second
        PauseTime = 0.25 ' Set duration.
        Start = Timer    ' Set start time.
        Do While Timer < Start + PauseTime
            DoEvents     ' Yield to other processes.
        Loop
    Next K

    Case Is = 1 'splitting piles
'reset FirstPileDone and SecondPileDone and QueryPile1 and
QueryPile2
    SecondPileDone = False
    QueryPile1 = 0
    QueryPile2 = 0
'discover how many multicard piles exist
    Call PileCount
'is this the last multicard pile?
    If MultiCardPileCount < 2 Then 'it's the first split AND
the last multicard pile (RARE)
'find piles to query highlight; identify first, QueryPile1
and second, QueryPile2
        For K = 0 To Ncards - 1
            If SortDesk!card(K).Enabled Then
                If  QueryPile1  =  0  Then  QueryPile1  =
Card2Pile(K, 0)
                If  (Card2Pile(K,  0)  <>  QueryPile1)  And
(QueryPile2 = 0) Then QueryPile2 = Card2Pile(K, 0)
            End If
        Next K
'lowlight other query pile
        PToggle = 4
        TogglePile (PileOrder(QueryPile2, 1))
'highlight query pile for this cycle
        PToggle = 1
        TogglePile (PileOrder(QueryPile1, 1))
        SortDesk!TaskText.Caption = "In the text box to the
right, please enter your explanation for why the highlighted
card            belongs            by            itself.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(2)
        Query(2).Top = Rationale.Top - Query(2).Height
        Query(2).Left = Rationale.Left
'show Query(2)
```

```
        Query(2).Visible = True
        Query(0).Visible = False
'erase the text
        Rationale.Text = ""
'show Rationale
        Rationale.Visible = True
'give cursor focus in Rationale
        Rationale.SetFocus
'show the Enter button
        TextDone.Visible = True
'set QueryMode = True
        QueryMode = True
        LastSplit = True
'flash the current task label
        Exit Sub
    End If

'**************************normal path for stage 1 starts
here*****************************
'find piles to query about; identify first, QueryPile1 and
second, QueryPile2
    For K = 0 To Ncards - 1
        If SortDesk!card(K).Enabled Then
            If QueryPile1 = 0 Then QueryPile1 = Card2Pile(K,
0)
            If   (Card2Pile(K,   0)   <>   QueryPile1)   And
(QueryPile2 = 0) Then QueryPile2 = Card2Pile(K, 0)
        End If
    Next K
'lowlight non-query pile
    PToggle = 4
    TogglePile (PileOrder(QueryPile2, 1))
'highlight query pile for this cycle
    PToggle = 1
    TogglePile (PileOrder(QueryPile1, 1))
'Set TaskText depending upon single or multi-card pile
    If PileOrder(QueryPile1, 0) < 3 Then
        SortDesk!TaskText.Caption = "In the text box to the
right, please enter your explanation for why the highlighted
cards            belongs            by            itself.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(2)
        Query(2).Top = Rationale.Top - Query(2).Height
        Query(2).Left = Rationale.Left
```

```
'show Query(2)
        Query(2).Visible = True
        Query(0).Visible = False
    Else:   SortDesk!TaskText.Caption = "In the text box to
the right, please enter your explanation for why the
highlighted cards belong together.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(0)
        Query(0).Top = Rationale.Top - Query(0).Height
        Query(0).Left = Rationale.Left
'show Query(0)
        Query(0).Visible = True
        Query(2).Visible = False
    End If
'erase the Rationale text
    Rationale.Text = ""
'show Rationale
    Rationale.Visible = True
'give cursor focus in Rationale
    Rationale.SetFocus
'show the Enter button
    TextDone.Visible = True
'set QueryMode = True
    QueryMode = True
'flash the current task label
'-------------------------------------------------
    Case Is = 2 'combine piles to unity
'test and trap for error conditions
'OnPIndex is last pile made
'lowlight everything
    PToggle = 4
    For K = 0 To Ncards - 1
        TogglePile (K)
    Next K
'highlight query pile for this cycle
    PToggle = 1
    TogglePile (PileOrder(OnPIndex, 1))
'set TaskText to multi-card pile query
    SortDesk!TaskText.Caption = "In the text box to the
right, please enter your explanation for why the highlighted
cards belong together.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
```

```
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(0)
    Query(0).Top = Rationale.Top - Query(0).Height
    Query(0).Left = Rationale.Left
'show Query(0)
    Query(0).Visible = True
    Query(2).Visible = False
'erase the text
    Rationale.Text = ""
'show Rationale
    Rationale.Visible = True
'give cursor focus in Rationale
    Rationale.SetFocus
'show the Enter button
    TextDone.Visible = True
'set QueryMode = True
    QueryMode = True
'-------------------------------------------------------
'finished with importance weighting mode
    Case Is = 5
'trap for cards remaining in deck
    For K = 0 To Ncards - 1
        If    (SortDesk!card(K).Left    =    DeckX)    And
(SortDesk!card(K).Top = DeckY) Then Exit Sub
    Next K
'on 'Done', disable 'em all
    For K = 0 To Ncards - 1
        SortDesk!card(K).Enabled = False
        SortDesk!card(K).Visible = False
    Next K
'compute & write the importance distances
'load DistOrder
    For K = 0 To Ncards - 1
        DistOrder(K) = K
    Next K
'sort 'em into order to find most and least important
    For I = Ncards - 1 To 0 Step -1
        For K = 0 To I
'test remaining cards for farther rightness,
            If          SortDesk!card(K).Left          >
SortDesk!card(DistOrder(I)).Left Then
'it's less important, so swap these two in the ImpOrder
array
                TDistOrd = DistOrder(I)
                DistOrder(I) = DistOrder(K)
                DistOrder(K) = TDistOrd
```

```
            End If
        Next K
    Next I
'compute maximum span
    Span  =  SortDesk!card(DistOrder(Ncards  -  1)).Left  -
SortDesk!card(DistOrder(0)).Left
'flag importance data and sort time
    Write #2, Now
'write normalized distances as a matrix
    For I = 0 To Ncards - 1
        For J = 0 To Ncards - 1
            Write #2, Abs(Span * (SortDesk!card(I).Left -
SortDesk!card(J).Left) / SortDesk.Width)
        Next J
    Next I
'set things up for difficulty ranking
    SortStage = 6
'put the lines in back
    For K = 0 To 3
        SortDesk!Line1(K).ZOrder 1
    Next
'change the LikertLine labels
    SortDesk!LikertLine(0).Caption = "Most Difficult"
    SortDesk!LikertLine(1).Caption = "Difficult"
    SortDesk!LikertLine(2).Caption = "Least Difficult"
    For K = 0 To 2
        SortDesk!Line1(K).Visible = True
        SortDesk!Line1(K).ZOrder 1
    Next K
'show the help
    SortDesk!TaskText.Caption = "Drag cards off the ""card
deck"" below and drop them from left (""Most Diffi- cult"")
to right (""Least Difficult"") accord- ing to difficulty.
If two or more cards have exactly the same difficulty, stack
them                                                vertically.
The ""Done"" button only works when you have moved all the
cards."
'    SortDesk!TaskText.Caption = "Drag cards off the ""card
deck"" below and drop them from left (""Most Impor- tant"")
to right (""Least Important"") ac- cording to importance.
If two or more cards have exactly the same importance, stack
them                                                vertically.
The ""Done"" button only works when you have moved all the
cards."
    SortDesk!TaskLabel.Visible = True
    SortDesk!Label1(3).Visible = True
    SortDesk!Label1(2).Visible = True
```

```
    SortDesk!TaskText.Visible = True
    SortDesk!Ok.Visible = True
    SortDesk!TaskLabel.ZOrder
    SortDesk!Label1(3).ZOrder
    SortDesk!Label1(2).ZOrder
    SortDesk!TaskText.ZOrder
    SortDesk!Ok.ZOrder
    TIndex = 1
'set                up                the                deck
About fifty sets will be presented and you are encouraged to
make snap judgments.  Just blast away..."
    Call Stacker
'set focus to the Ok button
    SortDesk!Ok.SetFocus
'flash the current task label
    For K = 1 To 20
        If SortDesk!TaskLabel.ForeColor = &H400000 Then
            SortDesk!TaskLabel.ForeColor = &HFFFFFF
        Else: SortDesk!TaskLabel.ForeColor = &H400000
        End If
'pause for ~1/4 second
        PauseTime = 0.25 ' Set duration.
        Start = Timer    ' Set start time.
        Do While Timer < Start + PauseTime
            DoEvents     ' Yield to other processes.
        Loop
    Next K
'-------------------------------------------------------
--
'done with difficulty ratings
    Case Is = 6
'trap for cards remaining in deck
    For K = 0 To Ncards - 1
        If     (SortDesk!card(K).Left     =     DeckX)     And
(SortDesk!card(K).Top = DeckY) Then Exit Sub
    Next K
'compute & write the difficulty distances
'load DistOrder
    For K = 0 To Ncards - 1
        DistOrder(K) = K
    Next K
'sort 'em into order to find most and least important
    For I = Ncards - 1 To 0 Step -1
        For K = 0 To I
'test remaining cards for farther rightness,
            If            SortDesk!card(K).Left            >
SortDesk!card(DistOrder(I)).Left Then
```

```
'it's less important, so swap these two in the ImpOrder
array
                TDistOrd = DistOrder(I)
                DistOrder(I) = DistOrder(K)
                DistOrder(K) = TDistOrd
            End If
        Next K
    Next I
'compute maximum span
    Span = SortDesk!card(DistOrder(Ncards - 1)).Left -
SortDesk!card(DistOrder(0)).Left
'flag importance data and sort time
    Write #2, Now
'write normalized distances as a matrix
    For I = 0 To Ncards - 1
        For J = 0 To Ncards - 1
            Write #2, Abs(Span * (SortDesk!card(I).Left -
SortDesk!card(J).Left) / SortDesk.Width)
        Next J
    Next I
'EXIT and write time & date to file
    Write #2, Now
'blank the cards
    For K = 0 To Ncards - 1
        SortDesk!card(K).Visible = False
    Next K
'blank the other stuff
    SortDesk!Ok.Visible = False
    SortDesk!Query(2).Visible = False
    SortDesk!Rationale.Visible = False
    SortDesk!TextDone.Visible = False
'turn off the ranking controls & lines
    SortDesk!DeckLabel.Visible = False
    For K = 0 To 2
        SortDesk!LikertLine(K).Visible = False
    Next K
    For K = 0 To 3
        SortDesk!Line1(K).Visible = False
    Next K
'flash thanks
    SortDesk!TaskText.Height = 3240
    SortDesk!TaskText.Width = 7215
    SortDesk!Label1(2).Height = 3240
    SortDesk!Label1(3).Height = 3240
    SortDesk!Label1(2).Width = 135
    SortDesk!Label1(3).Width = 135
    SortDesk!TaskLabel.Width = 7485
```

```
    SortDesk!TaskLabel.Height = 255
    SortDesk!TaskText.Move          (SortDesk.Width          -
SortDesk!TaskText.Width)  /  2  +  SortDesk!Label1(3).Width,
(SortDesk.Height   -   SortDesk!TaskText.Height)   /   2   +
SortDesk!TaskLabel.Height
    SortDesk!Label1(3).Move     SortDesk!TaskText.Left     -
SortDesk!Label1(3).Width, SortDesk!TaskText.Top
    SortDesk!Label1(2).Move     SortDesk!TaskText.Left     +
SortDesk!TaskText.Width, SortDesk!TaskText.Top
    SortDesk!TaskLabel.Move         SortDesk!Label1(3).Left,
SortDesk!TaskText.Top - SortDesk!TaskLabel.Height
    SortDesk!TaskText.Font.Size = 20
    SortDesk!TaskText.Caption = "                    Thanks for
the                                                     help...
Please wait for the a: drive                          to finish
writing to the disk                 and remember to drop off
your disk              on Wednesday or Thursday"
'    SortDesk!TaskText.Caption = "                    Thanks for
the                                                     help...
Please wait for the a: drive                          to finish
writing to the disk                 and remember to drop off
your disk                          on Wednesday"
    SortDesk!TaskText.Visible = True
    SortDesk!TaskLabel.Visible = True
    SortDesk!Label1(3).Visible = True
    SortDesk!Label1(2).Visible = True
    SortDesk!Label1(3).ZOrder
    SortDesk!Label1(2).ZOrder
    SortDesk!TaskText.ZOrder

'pause for 3 seconds
    PauseTime = 6 ' Set duration.
    Start = Timer   ' Set start time.
    Do While Timer < Start + PauseTime
        DoEvents    ' Yield to other processes.
    Loop
    End
    End Select
End Sub


Private Sub Done_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
```

```
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub


Private Sub Done_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon     =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub Exit_Click()
Write #2, Now
End
End Sub
```

```
Private Sub Exit_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub


Private Sub Exit_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon     =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select
```

```
End Sub


Private Sub Form_Click()
'exit if nothing highlighted
    If PileMode = False Then Exit Sub
'reset the HighPile to PostIt
    PToggle = 1
    TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
    HighPile = -1
'reset PileMode to false
    PileMode = False
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y
As Single)
'which pile did this come from?
    OldPIndex = Card2Pile(Source.Index, 0)
'determine move coordinates
    DelX = X - DragX - OldLeft
    DelY = Y - DragY - OldTop
    Select Case PileMode
'if in PileMode, move and repaint as highlighted
    Case Is = True
        For K = 1 To PileOrder(OldPIndex, 0) - 1
            SortDesk!card(PileOrder(OldPIndex,      K)).Move
SortDesk!card(PileOrder(OldPIndex,    K)).Left    +    DelX,
SortDesk!card(PileOrder(OldPIndex, K)).Top + DelY
        Next K
        PToggle = 0
        TogglePile (Source.Index)
'if not in PileMode, take the card from its pile, start a
new one, move, and repaint the card as lowlighted
    Case Is = False
'take the card from the old pile
        Take1FromPile (Source.Index)
'start a new pile for it on SortDesk
        StartPile (Source.Index)
'move the card and make visible as card
        SortDesk!card(Source.Index).Move
SortDesk!card(Source.Index).Left          +          DelX,
SortDesk!card(Source.Index).Top + DelY
        PToggle = 1
        TogglePile (Source.Index)
'tidy the old pile if appropriate
        'ShrinkPile (OldPIndex)
```

```
    End Select


'track pile changes with temporary windows
    'SortDesk!PIO(0).Caption = Card2Pile(Source.Index, 0)
    'SortDesk!PIO(1).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 0)
    'SortDesk!PIO(2).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 1)
    'SortDesk!PIO(3).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 2)
    'SortDesk!PIO(4).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 3)
    'SortDesk!PIO(6).Caption                                    =
PileOrder(Card2Pile(Source.Index, 0), 4)
End Sub



Private Sub Help_Click()
'kill the pile highlight, if any
    If PileMode = True Then
'reset the HighPile to PostIt
        PToggle = 1
        TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
        HighPile = -1
'reset PileMode to false
        PileMode = False
    End If
'show help
    Label1(2).Visible = True
    Label1(2).ZOrder
    Label1(3).Visible = True
    Label1(3).ZOrder
    TaskText.Visible = True
    TaskText.ZOrder
    TaskLabel.Visible = True
    TaskLabel.ZOrder
    SortDesk!Ok.Visible = True
    SortDesk!Ok.ZOrder
    If QueryMode = False Then SortDesk!Ok.SetFocus
'set IGotIt to False
    IGotIt = False

End Sub
```

```
Private Sub Help_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub


Private Sub Help_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon     =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select
```

```
End Sub


Private Sub IOBar_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub IOBar_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon     =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
```

```
    End Select

End Sub



Private Sub PileZeroBar_DragDrop(Source As Control, X As
Single, Y As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub PileZeroBar_DragOver(Source As Control, X As
Single, Y As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over PilZeroBar; set DragIcon to "illegal drag
area icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving PileZeroBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
```

```
        SortDesk!card(Source.Index).DragIcon                =
SortDesk!PIO(0).DragIcon
        SortDesk!card(Source.Index).DragIcon                =
SortDesk!PIO(1).DragIcon
    End If

    Case OVER
    End Select

End Sub


Private Sub Ok_Click()
'turn off the help stuff
    Label1(2).Visible = False
    Label1(3).Visible = False
    TaskText.Visible = False
    TaskLabel.Visible = False
    SortDesk!Ok.Visible = False
'set IGotIt to true
    IGotIt = True

End Sub

Private Sub Query_DragDrop(Index As Integer, Source As
Control, X As Single, Y As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub Query_DragOver(Index As Integer, Source As
Control, X As Single, Y As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
```

```
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:           SortDesk!card(Source.Index).DragIcon    =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub Rationale_DragDrop(Source As Control, X As
Single, Y As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub Rationale_DragOver(Source As Control, X As
Single, Y As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
```

```
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:           SortDesk!card(Source.Index).DragIcon    =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub TaskLabel_DragDrop(Source As Control, X As
Single, Y As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub TaskLabel_DragOver(Source As Control, X As
Single, Y As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
```

```
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon     =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub TaskText_DragDrop(Source As Control, X As
Single, Y As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub TaskText_DragOver(Source As Control, X As
Single, Y As Single, State As Integer)
```

```
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:           SortDesk!card(Source.Index).DragIcon    =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub TextDone_Click()
'flush text to appropriate file
'unset QueryMode
'kill the pile highlight, if any
    If PileMode = True Then
'reset the HighPile to PostIt
        PToggle = 1
        TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
        HighPile = -1
'reset PileMode to false
        PileMode = False
    End If

    Select Case SortStage

    Case Is = 0 'pile title and write stage; occurs after
"Done"
```

```
'write QueryPile to file if not empty
    If QueryPile > 0 Then WritePile (QueryPile)
'write Rationale text to file (may be empty)
    If Rationale.Text <> "" Then Write #2, Rationale.Text
'empty Rationale of text
    Rationale.Text = ""
'return focus to Rationale
    Rationale.SetFocus
'disable all the cards
    PToggle = 4
    For K = 0 To Ncards - 1
        TogglePile (K)
    Next K
'query hightlight the next pile if we haven't run out of
piles
    QueryPile = NextQPile()
    If QueryPile = 0 Then 'on finding "end of piles,"
advance to importance ranking
        SortStage = 5
        QueryMode = False
'turn off all the sorting controls
        For K = 0 To Ncards - 1
            SortDesk!card(K).Visible = False
        Next K
        SortDesk!Query(2).Visible = False
        SortDesk!Rationale.Visible = False
        SortDesk!TextDone.Visible = False
        SortDesk!IOBar.Visible = False
'turn on the ranking controls & put the lines in back
        SortDesk!DeckLabel.Visible = True
        For K = 0 To 2
            SortDesk!LikertLine(K).Visible = True
        Next K
        For K = 0 To 3
            SortDesk!Linel(K).Visible = True
            SortDesk!Linel(K).ZOrder 1
        Next K
'show the help
        SortDesk!TaskText.Caption = "Drag cards off the
""card deck"" below and drop them from left (""Most Impor-
tant"") to right (""Least Important"") ac- cording to
importance.
If two or more cards have exactly the same importance, stack
them                                                vertically.
The ""Done"" button only works when you have moved all the
cards."
        SortDesk!TaskLabel.Visible = True
```

```
        SortDesk!Label1(3).Visible = True
        SortDesk!Label1(2).Visible = True
        SortDesk!TaskText.Visible = True
        SortDesk!Ok.Visible = True
        SortDesk!Ok.SetFocus
        SortDesk!TaskLabel.ZOrder
        SortDesk!Label1(3).ZOrder
        SortDesk!Label1(2).ZOrder
        SortDesk!TaskText.ZOrder
        SortDesk!Ok.ZOrder
        TIndex = 1
'do          the          importance          weighting/ranking
About fifty sets will be presented and you are encouraged to
make snap judgments.  Just blast away..."
        Call Stacker
'set focus to the Help button
        SortDesk!Help.SetFocus
'flash the current task label
        For K = 1 To 20
            If SortDesk!TaskLabel.ForeColor = &H400000 Then
                SortDesk!TaskLabel.ForeColor = &HFFFFFF
            Else: SortDesk!TaskLabel.ForeColor = &H400000
            End If
'pause for ~1/4 second
            PauseTime = 0.25 ' Set duration.
            Start = Timer    ' Set start time.
            Do While Timer < Start + PauseTime
                DoEvents     ' Yield to other processes.
            Loop
        Next K
        Exit Sub
    End If

'normal entry  stage 0*****************************
    PToggle = 1
    TogglePile (PileOrder(QueryPile, 1))

'Set TaskText depending upon single or multi-card pile
    If PileOrder(QueryPile, 0) < 3 Then
        SortDesk!TaskText.Caption = "In the text box below,
please enter a title for the highlighted single-card pile.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Title"" button at the
right of the text box."
'move Query(2)
'       Query(2).Top = Rationale.Top - Query(2).Height
```

```
'        Query(2).Left = Rationale.Left
'show Query(2)
       Query(2).Visible = True
'        Query(0).Visible = False
    Else:   SortDesk!TaskText.Caption = "In the text box
below, please enter a title for the highlighted multi-card
pile.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(0)
'        Query(0).Top = Rationale.Top - Query(2).Height
'        Query(0).Left = Rationale.Left
'show Query(2)
'        Query(0).Visible = True
       Query(2).Visible = True
    End If
'show Rationale
    Rationale.Visible = True
'give cursor focus in Rationale
    Rationale.SetFocus
'show the Enter button
    TextDone.Visible = True
'set QueryMode = True
    QueryMode = True
'flash the current task label

    Case Is = 1 'splitting piles
    If SecondPileDone = False Then 'first split rationale
TextDone; set up second
'write QueryPile1 to file
        WritePile (QueryPile1)
'write Rationale text to file
        Write #2, Rationale.Text
'low light the old QueryPile1
        PToggle = 4
        TogglePile (PileOrder(QueryPile1, 1))
'highlight query pile for this part of split cycle
        PToggle = 1
        TogglePile (PileOrder(QueryPile2, 1))
'set TaskText depending upon single or multi-card pile
        If PileOrder(QueryPile2, 0) < 3 Then
            SortDesk!TaskText.Caption = "In the text box to
the right, please enter your explanation for why the
highlighted cards belongs by itself.
Don't worry about speling or tpyos, just blaze away.  QUICK
```

```
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(2)
            Query(2).Top = Rationale.Top - Query(2).Height
            Query(2).Left = Rationale.Left
'show Query(2)
            Query(2).Visible = True
            Query(0).Visible = False
        Else:   SortDesk!TaskText.Caption = "In the text box
to the right, please enter your explanation for why the
highlighted          cards          belong          together.
Don't worry about speling or tpyos, just blaze away.  QUICK
TIP:  clicking on a card inserts its text at the cursor.
When you finish, click on the ""Enter Text"" button at the
right of the text box."
'move Query(0)
            Query(0).Top = Rationale.Top - Query(0).Height
            Query(0).Left = Rationale.Left
'show Query(0)
            Query(0).Visible = True
            Query(2).Visible = False
        End If
'erase the Rationale text
        Rationale.Text = ""
'show Rationale
        Rationale.Visible = True
'give cursor focus in Rationale
        Rationale.SetFocus
'show the Enter button
        TextDone.Visible = True
'set QueryMode to True
        QueryMode = True
        SecondPileDone = True
'flash the current task label
        Exit Sub
    End If

'*************************************************************
'entry if SecondPileDone = True
'write QueryPile2 to file
    WritePile (QueryPile2)
'write Rationale text to file
    Write #2, Rationale.Text
'hide Query(0)& Rationale
        Query(0).Visible = False
        Query(2).Visible = False
```

```
        Rationale.Visible = False
        TextDone.Visible = False
        QueryMode = False
        SplitIDd = False
'lowlight all the cards
    PToggle = 4
    For K = 0 To Ncards - 1
        TogglePile (K)
    Next K

'find out how many multicard piles remain
    Call PileCount

'if finished with splitting, split ******************end of
pile sort
    If MultiCardPileCount < 1 Then
'write blank line and date and time to file
        Write #2,
        Write #2, Now
'flash THANKS and exit
        SortStage = 4
'blank the query and textbox and textdone button
'hide Query(0)& Rationale
        QueryMode = False
        Query(0).Visible = False
        Query(2).Visible = False
        Rationale.Visible = False
        TextDone.Visible = False

'set TaskText to triad test instructions
        SortDesk!TaskText.Caption = "Decide which one of the
three cards ""belongs least"" and then click on it.
SortDesk will present about fifty sets of three cards to you
and then automatic- ally take you to the next task upon
completion of this one."
        For K = 0 To Ncards - 1
            SortDesk!card(K).Visible = False
            SortDesk!card(K).Enabled = False
        Next K
        SortDesk!card(PileOrder(0, 1)).Move 1200, 2280
        SortDesk!card(PileOrder(0, 2)).Move 4680, 2280
        SortDesk!card(PileOrder(0, 3)).Move 8160, 2280
        For K = 1 To 3
            SortDesk!card(PileOrder(0, K)).Width = 2535
            SortDesk!card(PileOrder(0, K)).Height = 1335
            SortDesk!card(PileOrder(0, K)).Font.Size = 18
            SortDesk!card(PileOrder(0, K)).Visible = True
```

```
            SortDesk!card(PileOrder(0, K)).Enabled = True
            SortDesk!card(PileOrder(0,    K)).BackColor    =
&HC0FFFF 'card Yellow
            SortDesk!card(PileOrder(0,    K)).ForeColor    =
&H800000 'text/pile Blue
            SortDesk!card(PileOrder(0, K)).BorderStyle = 1
            SortDesk!card(PileOrder(0, K)).BackStyle = 1
         Next K
         QueryMode = False
        Exit Sub
    End If

'otherwise,              reset              for              next
split*********************************

'set QueryPile1 and QueryPile2 to 0
    QueryPile1 = 0
    QueryPile2 = 0
'set SecondPileDone to False
    SecondPileDone = False
    PToggle = 1
    If MultiCardPileCount = 1 Then
'set TaskText to appropriate stage 1 task
        SortDesk!TaskText.Caption = "Decide how to split the
remaining multi- card pile into two ""chunks"" that belong
together   least.      (Chunks   can   be   any   size.)
Then, double click on the pile and divide it into those two
chunks.
When you finish, click the ""Done"" but- ton on the far
right."
'query highlight the remaining multicard pile
        TogglePile (PileOrder(MultiCardPile(1), 1))
    Else:   SortDesk!TaskText.Caption = "Decide which one of
these" + Str(MultiCardPileCount) + " piles can be split into
two ""chunks"" which seem most dissimilar.  (Chunks of any
size.)
Then, double-click on that pile and split it into those two
chunks.    Split  ONLY  one  pile  into  ONLY  two  chunks.
When finished, click the ""Done"" button."
'query hightlight remaining multicard piles
        For K = 1 To MultiCardPileCount
            TogglePile (PileOrder(MultiCardPile(K), 1))
        Next K
'flash the current task label
    End If

    Case Is = 2 'if SortStage = 2 then
```

```
'turn on cards
        PToggle = 1
        For K = 0 To Ncards - 1
            TogglePile (K)
        Next K
'write OnPIndex pile to file
    WritePile (OnPIndex)
'write rationale text to file
    Write #2, Rationale.Text
'turn off Rationale, Query, and set QueryMode to False
        QueryMode = False
        Query(0).Visible = False
        Query(2).Visible = False
        Rationale.Visible = False
        TextDone.Visible = False
'set TaskText to SortStage 2 message
        Call PileCount
        If NPiles < 2 Then 'we're done with combining; shift
to splitting
'put a blank line in file for SS1
            Write #2,
            SortStage = 1
'set QueryPile1 and QueryPile2 to 0
            QueryPile1 = 0
            QueryPile2 = 0
'restore free sort display
            For K = 0 To Ncards - 1
                SortDesk!card(K).Move FSLeft(K), FSTop(K)
                Card2Pile(K, 0) = FSC2P(K, 0)
                Card2Pile(K, 1) = FSC2P(K, 1)
                Card2Pile(K, 2) = FSC2P(K, 2)
            Next K
'load Pile array
            For I = 0 To 49 Step 1
                For J = -14 To 14 Step 1
                    For K = -14 To 14 Step 1
                        Pile(I, J, K) = FSPile(I, J, K)
                    Next K
                Next J
            Next I
'load PileOrder array
            For M = 0 To 49
                For N = 0 To 49
                    PileOrder(M, N) = FSPO(M, N)
                Next N
            Next M
        QueryPile1 = 0
```

```
        QueryPile2 = 0
'disable all the cards
        PToggle = 4
        For K = 0 To Ncards - 1
            Call TogglePile(K)
        Next K
        QueryMode = False
        SortDesk!Label1(0).BackColor = &H40&
        SortDesk!Label1(2).BackColor = &H40&
        SortDesk!Label1(3).BackColor = &H40&
        SortDesk!TaskLabel.BackColor = &H40&
        SortDesk!TaskText.BackColor = &H40&

        Call PileCount
        PToggle = 1
        If MultiCardPileCount = 1 Then
'set TaskText to appropriate stage 1 task
            SortDesk!TaskText.Caption = "Decide how to split
the remaining multi- card pile into two ""chunks"" that
belong together least.    (Chunks can be any size.)
Then, double click on the pile and divide it into those two
chunks.
When you finish, click the ""Done"" but- ton on the far
right."
'query highlight the remaining multicard pile
            TogglePile (PileOrder(MultiCardPile(1), 1))
        Else:    SortDesk!TaskText.Caption = "Decide which
one of these" + Str(MultiCardPileCount) + " piles can be
split into two ""chunks"" which seem most dissimilar.
(Chunks              of              any              size.)
Then, double-click on that pile and split it into those two
chunks.    Split ONLY one pile into ONLY two chunks.
When finished, click the ""Done"" button."
'query highlight remaining multicard piles
            For K = 1 To MultiCardPileCount
                TogglePile (PileOrder(MultiCardPile(K), 1))
            Next K
        End If
    SecondPileDone = False
'flash the current task label
        For K = 1 To 20
            If        SortDesk!TaskLabel.ForeColor        =
SortDesk!TaskLabel.BackColor Then
                SortDesk!TaskLabel.ForeColor = &HFFFFFF
            Else:       SortDesk!TaskLabel.ForeColor       =
SortDesk!TaskLabel.BackColor
            End If
```

```
'pause for ~1/4 second
            PauseTime = 0.25 ' Set duration.
            Start = Timer   ' Set start time.
            Do While Timer < Start + PauseTime
                DoEvents    ' Yield to other processes.
            Loop
        Next K
        Exit Sub
    End If

'set TaskText to SortStage 2 message: normal entry to
2********************
    Call PileCount
    If NPiles < 3 Then
            SortDesk!TaskText.Caption = "Make one pile out
of the last two piles by dragging and dropping one on the
other.
When finished, click the ""Done"" button."
        Else
            SortDesk!TaskText.Caption = "Decide which 2 of
these" + Str(NPiles) + " piles belong together the most.
Then, make one pile out of those two by dragging and
dropping one on the other.
Combine ONLY two piles and take no existing pile apart
When finished, click the ""Done"" button."
    End If
'flash the current task label
End Select

End Sub

Public Sub PileCount()
dummy = dummy + 1
    NPiles = 0
    MultiCardPileCount = 0
    For K = 1 To 49
        If PileOrder(K, 1) > -1 Then NPiles = NPiles + 1
        If PileOrder(K, 0) > 2 Then
            MultiCardPileCount = MultiCardPileCount + 1
            MultiCardPile(MultiCardPileCount) = K
        End If
    Next K
End Sub

Private Sub TextDone_DragDrop(Source As Control, X As
Single, Y As Single)
'do Recovery Event:
```

```
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub TextDone_DragOver(Source As Control, X As
Single, Y As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon        =
SortDesk!PIO(0).DragIcon
    Else:          SortDesk!card(Source.Index).DragIcon  =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub Tidy_Click()
'kill the pile highlight, if any
```

```
    If PileMode = True Then
'reset the HighPile to PostIt
        PToggle = 1
        TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
        HighPile = -1
'reset PileMode to false
        PileMode = False
    End If
End Sub

Private Sub Tidy_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub Tidy_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
```

```
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon           =
SortDesk!PIO(0).DragIcon
    Else:           SortDesk!card(Source.Index).DragIcon    =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Private Sub Undo_Click()
'kill the pile highlight, if any
    If PileMode = True Then
'reset the HighPile to PostIt
        PToggle = 1
        TogglePile (PileOrder(HighPile, 1))
'reset HighPile to no pile
        HighPile = -1
'reset PileMode to false
        PileMode = False
    End If
End Sub

Private Sub Undo_DragDrop(Source As Control, X As Single, Y
As Single)
'do Recovery Event:
        Select Case PileMode
'if in Pile mode, restore flying Pile as highlighted, On
Pile as lowlighted if different
        Case Is = True
            PToggle = 0
            TogglePile (Source.Index)
'if not in Pile mode, restore the card(s) as lowlighted
        Case Is = False
            PToggle = 1
            TogglePile (Source.Index)
            TogglePile (Index)
        End Select

End Sub

Private Sub Undo_DragOver(Source As Control, X As Single, Y
As Single, State As Integer)
    DropPIndex = Card2Pile(Source.Index, 0)
'set the constants
```

```
    Const ENTER = 0
    Const LEAVE = 1
    Const OVER = 2
'ENTERing, LEAVEing, or OVER?
    Select Case State

    Case ENTER
'no entry over IOBar; set DragIcon to "illegal drag area
icon"
    SortDesk!card(Source.Index).DragIcon                    =
SortDesk!PIO(2).DragIcon

    Case LEAVE
'leaving IOBar; set DragIcon to pile or card DragIcon
    If PileMode = True Then
            SortDesk!card(Source.Index).DragIcon            =
SortDesk!PIO(0).DragIcon
    Else:           SortDesk!card(Source.Index).DragIcon    =
SortDesk!PIO(1).DragIcon
    End If
    Case OVER
    End Select

End Sub


Public Function NextQPile()
Static StartIndex
    StartIndex = StartIndex + 1
'run the array to find next query pile
    For K = StartIndex To 60
        If PileOrder(K, 0) > 1 Then
            NextQPile = K
            StartIndex = K
            Exit Function
        End If
    Next K
'do some error trapping maybe
    NextQPile = 0
End Function
```

VITA

Jeffrey A. Morrow

University of Washington

1998

Publications and Presentation Papers

Tracking Consensus in Product Development Teams, unpublished doctoral dissertation, June 1998.

"SortDesk: Pile Sorting, Magnitude Scaling and Triads Tests on a PC," *Cultural Anthropology Methods*, in review.

"Applications of Anthropology to Product Development," with Joseph A. Heim, in preparation.

"Modelling Product Development Processes," with Rob Smith, INFORMS Spring Conference, May 1997.

"Sustainable Competitive Advantage: How QFD Can Work," American Concurrent Engineering Conference, October 1996.

"Tracking Consensus Trajectories in Virtual Enterprise Product Development," IEEE Conference on Virtual Enterprises, August 1996.

"Using PRO/Engineer to Optimize Shapes of Physical Objects," Second Annual PRO/User's Group Conference, June 1991.

"Automated Finite Element Method Parameter Design Experimentation: A Steering Wheel Example," Design Producti-vity International Conference, February 1991.

"Quality Engineering Implementation in the United States," *The ASI Journal*, Volume 2: Number 2, Fall 1989.

"Quality Engineering in the United States," (edited and translated by Shin Taguchi), *The Journal of the Japan Standards Association*, May 1989

Education

UNIVERSITY of WASHINGTON — Seattle, Washington
Ph. D. Mechanical Engineering — June 1998

MIT SLOAN SCHOOL of MANAGEMENT — Cambridge, Massachusetts
S. M., Management of Technology — June 1988

- Thesis, "An Investigation of the Penetration of Taguchi Quality Engineering in Five U. S. Companies," was guided by Dr. Don P. Clausing and Dr. Michael A. Cusumano.
- Selected by Dr. Madhav S. Phadke to receive Xerox Award Scholarship.

UNIVERSITY of MINNESOTA — Minneapolis, Minnesota
B. Me. (Mechanical Engineering) — May 1981

Work Experience

ALLIEDSIGNAL Electronic Systems — Redmond, Washington
Organizational Effectiveness Specialist — 1995 to present

- Led group of senior engineers and managers to develop a product line architecture in anticipation of introduction of new technology. The new line achieves market segment variety via end-stage customization of standard submodules.
- Used robust design to dramatically improve laser cutting of fused silica and to redesign snap action thermal switches to obviate five painstaking adjustment/readjustment steps.
- Consult to senior management on TQM, lead process improvement teams, and develop performance support materials and learning systems.

BOEING COMMERCIAL AIRPLANE GROUP — Seattle, Washington
Manager, Management Development Curriculum Development — 1992 to 1994

- After senior management adopted Continuous Quality Improvement (CQI) as *the* management system, was given responsibility for both management development and CQI curricula. Formed and cultivated a staff of five performance technologists/instructional materials producers.

- Supervised the production of performance-based learning systems for leaders of process improvement teams, users of the "7 Management & Planning Tools," leaders of "5S Campaigns," and a general "Understanding Variation in Process Improvement" curriculum spanning the topics of SPC, "Statistical Thinking," and robust design.

- Led a high-visibility project to bring systems modeling to management education via "management flight simulators" and the use of **ithink**™ modeling software. Proficient at modeling with **ithink**™.

Manager, Continuous Quality Improvement Training — 1989 to 1992

- Grew a group of eight highly individual consultant/instructors into a largely self-managing team of twenty staff members and twenty-five contractors.

- Led the delivery of 410,000 student-hours of CQI training in 1991, almost quadrupling the team's 1990 delivery. 1992 delivery was nearly 450,000 student-hours at completion.

- Managed a budget of $2.1 million.

- Wrote and delivered two technical papers on computer automation of robust design for products whose geometry drives optimal design. Convinced two external (to Boeing) value-added resellers to donate expertise, equipment, and software to support development of the software needed for the project.

ORR:LOESS — Seattle, Washington
Principal — 1988 to present

- Consulted and trained in total quality management for U. S. firms in the West and Southwest. For one high-tech forging firm, gained commitment of senior management and trained employees for successful roll-out of a six-team process

improvement program built on robust design and teamwork between hourly workers and professional staff.

- Subcontracted consulting and training for the American Supplier Institute, primarily in robust design, SPC, and Quality Function Deployment (QFD).

GE AIRCRAFT ENGINES BUSINESS GROUP — Evendale, Ohio
Engineering Systems Analyst — 1985 to 1987

- Wrote the specification document for the Mechanical Design Engineering Workstation. Captured the requirements of the engineers with focus groups and storyboarding. Researched state of the art in CAD/CAM, finite element analysis, workstation hardware, computer networking, daily work management, etc. (1985-1986)

- Vectorized and otherwise updated graphical pre- and postprocessor computational fluid dynamics software for use with a Cray XMP 2-8 supercomputer and Apollo workstations. (1986-1987)

- Led the Cray User's Group for all of GE; founded and sustained the Taguchi Method Study Group.

CONTROL DATA CORPORATION — Roseville, Minnesota
Engineering Applications Analyst — 1981 to 1985

- Principal support analyst for CYBERNET time sharing service. Assisted client engineers world-wide in use of engineering design, analysis, and optimization applications as follows:

- Large-scale energy analysis of buildings and thermal and nuclear power plants; optimization of unit commitment and economic dispatch for electrical power systems (1981-1983);

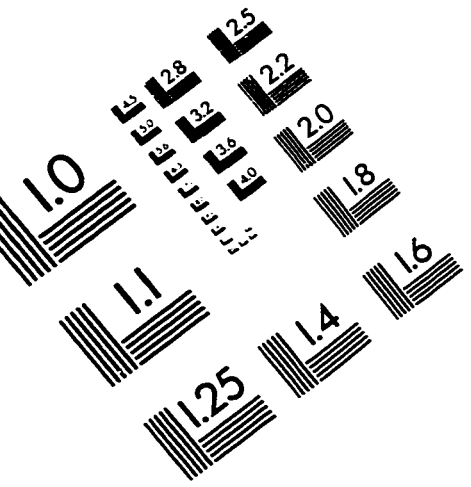
- Finite element analysis of structures, magnetics, and fluid flow (1983-1984);

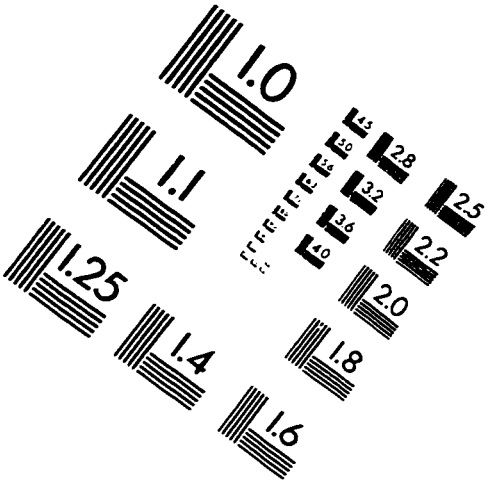
- Computer-aided design (for mechanical and electronic products), computer-aided manufacturing (mostly machining and mold design), solids modeling, and engineering database design and product configuration management (1984-1985).

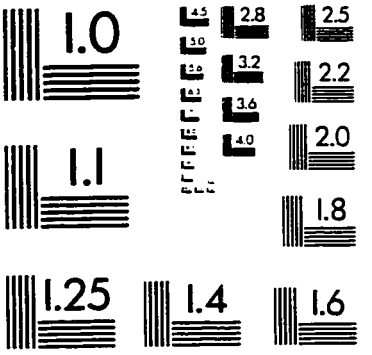
Affiliations

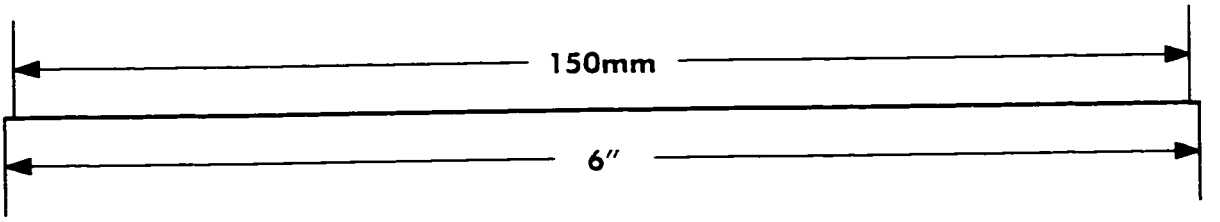
American Society of Mechanical Engineers
American Society for Quality Control
International Society for Performance & Instruction
Society of Manufacturing Engineers
INFORMS
System Dynamics Society

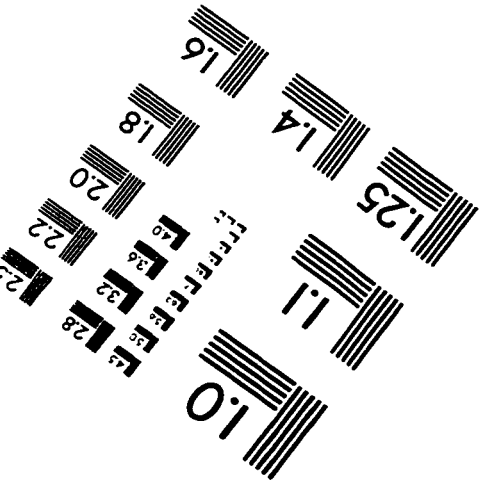
IMAGE EVALUATION
TEST TARGET (QA-3)

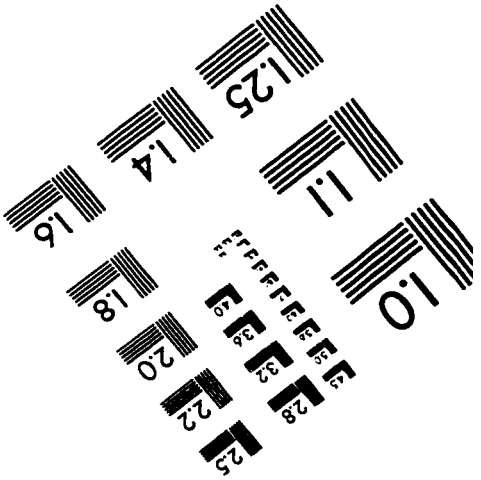












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989