Communications of the Association for Information Systems

Volume 20 Article 21

October 2007

An Empirical Assessment of User Perceptions of Feature versus Application Level Usage

Michael J. Harrison
Carnegie Mellon University

Pratim Datta
Louisiana State University, pdatta2@lsu.edu

Recommended Citation
Harrison, Michael J. and Datta, Pratim (2007) "An Empirical Assessment of User Perceptions of Feature versus Application Level Usage," *Communications of the Association for Information Systems*: Vol. 20 , Article 21.
DOI: 10.17705/1CAIS.02021
Available at: https://aisel.aisnet.org/cais/vol20/iss1/21

Communications of the Association for Information Systems

AN EMPIRICAL ASSESSMENT OF USER PERCEPTIONS OF FEATURE VERSUS APPLICATION LEVEL USAGE

Michael J. Harrison
Heinz School of Public Policy and Management
Carnegie Mellon University

Pratim Datta
Department of Management and Information Systems
Kent State University
pdatta@kent.edu

ABSTRACT

Users often use application software because of particular features. However, little remains known on whether user perceptions of application use is merely *feature-driven* or whether users perceive *their application use as being more than an amalgamation of features*. As the software industry ushers trends such as Web services, it becomes evermore important for vendors and users alike to clarify how users perceive features and applications. The paper is an attempt to confirm whether users can perceptively unbundle application software features from the overall applications themselves. Using a modified version of the repertory grid technique, this study investigates user perceptions of application features using data collected from users in the design and development departments across five firms. The results suggest that *user perceptions of overall applications overshadow their perceptions of independent features,* suggesting application-level lock-in effects and pointing out the difficulty in vendor attempts to unbundle features from feature categories and applications. The study closes with a discussion of the findings and offering cues for future research.

Keywords: application features, personal construct theory, modified repertory grid technique

I. INTRODUCTION

The tremendous innovation in application software has primarily been feature-driven. Look at any application portfolio in an organization, and you will find a variety of application suites (e.g. graphics, development, desktop) containing multiple licenses of applications redundant in feature categories and features. Applications contain features that offer users predefined functionality and control in their use. Features are vendor-created software tools designed to complete tasks on behalf of the user. In applications, features are grouped within feature categories. An application, in turn, is a bundle of features compiled to provide users with the ability to complete a set of specialized tasks. A feature, therefore, is a software tool enveloped as a function or routine (e.g. debuggers in programming applications; data sorting in spreadsheet and database applications; layers in image manipulation applications; search tools in browsers and processors). However, for the purposes of this study, features do not include application characteristics such as interface design and memory utilization. For example, the format feature *category* in the

Microsoft Word *application* consists of *features* related to the formatting of styles, fonts, paragraphs, among others.

Regardless of the type of software application, inter-vendor competition as well as iterative developments and upgrades have led to a considerable amount of feature overlap both within and across applications. Moreover, organizations spend considerable sums of money investing in similar types of applications, sometimes just for the sake of a meager set of features demanded by certain users. Consequently, an organization's application software portfolio often includes multiple applications with considerable overlap of features. In fact, recent trends in software asset management (SAM) stress on reducing "shelfware"- applications licensed (adopted) but not used because of available alternatives sharing a set of "commonly used" "redundant" features. For example, Adobe Photoshop Elements and Corel PhotoDraw share a host of common features; Corel WordPerfect and Microsoft Word similarly share a high degree of application-level feature overlap. Figure 1 shows the hierarchy within applications.

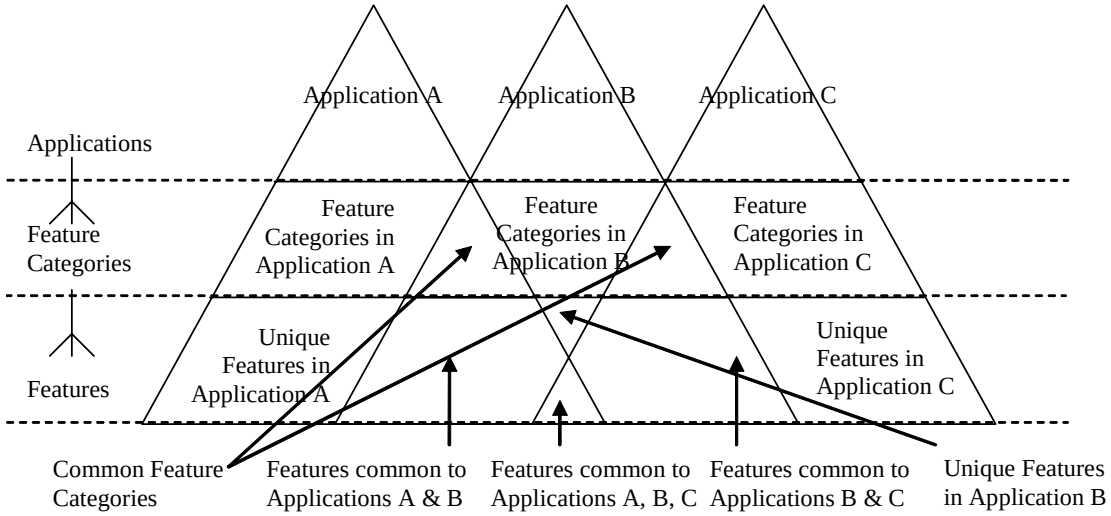

Figure 1. Nested Hierarchies in Applications

In software asset management, a common assumption made is that users are rational. As rational beings, users are likely to be feature-driven. Using such a contention, users would choose an application because of its inherent features and should be indifferent toward any other application containing the similar feature set, unless, of course, users perceive an application as more than a sum of its features. Interestingly, while overall technology acceptance has gained prominence in IS research, [Davis 1989; Thompson et al. 1991; Moore and Benbasat 1991; Compeau and Higgins 1995], an examination of feature-driven application usage has escaped scrutiny [Zmud et al. 2004]. Little exists in previous literature regarding feature elicitation among categories of applications, thus failing to address the issue of application dependence in feature level usage. Indeed, "prior studies have invariably examined IT applications as a whole (i.e., as a 'black box') rather than focusing on the specific feature sets that comprise each application's functionality" [Zmud et al. 2004: 2].

While it is true that every application offers some unique features, customers have to pay a considerable premium for buying the entire application software in order to use the few features offered in a new version or as an upgrade. Unbundling features from an application, although difficult, is being promoted by the next generation of application software. Microsoft .Net and J2EE environments are promoting a new generation of application-independent features called Web services that can be invoked and consumed (used) over the Web. For example, instead of

resorting to a resource-intensive application such as a Adobe Photoshop, a firm planning to "batch resize" their images could instead search online for a "batch resizing" feature, listed as a Web service, and use it on an ad hoc basis without being mired by issues of platform and application dependencies. By finding and consuming openly available features, firms can add significant efficiencies by moving away from application installation and maintenance. Google, for instance, offers a variety of search and map APIs (Application Programming Interface) for enterprise use of search and mapping features without having to install and maintain large stand-alone GIS (Geographical Information Systems) and search applications.

The driving motivation behind understanding application and feature level use is a concerted effort toward reducing application underutilization, a common problem where organizations end up using less than 10 percent of an application's entire set of features while paying site license fees for the entire application. The issue becomes even more acute as users within an organization request multiple "similar" applications, led by the choice of a very few features, leading to non-value added costs from shelfware. For example, a user may want to use SPSS to conduct missing data analysis, Minitab to chart the data, and SAS Base to run a regression model. Noticeably, the user only utilizes a handful of features in each of the three types of application software, leading to application underutilization. After all, organizations well realize that consolidating popular features can offer considerable savings. Therefore, understanding feature-driven application usage becomes an important aspect in IS planning: from feature-driven application customization by developers to feature consolidation strategies by vendors and IS management.

In the face of relevant organizational concerns, this research seeks to explain user perceptions toward feature and application level usages, under conditions of feature overlaps and redundancies in a common application portfolio. Until now, researchers have primarily investigated applications from the context of systems adoption and use. However, such an approach does not distinguish the hierarchy of usage: from applications to feature categories to individual features. As such, this study investigates whether users can perceptively distinguish features and feature categories from their parent applications, thus offering a relatively detailed understanding of application and feature-level use among organizational users. Even if users use all aspects of an application equally or just a few features, the question is, *can users perceptively unbundle specific features from the overarching feature categories and the application?* Specifically, we investigate whether user perceptions of application features are independent of the application type and feature categories.

The paper makes three principal contributions: First, if it is found that users are indeed rational and can separate underlying features from an application, firms can unbundle features from applications to consolidate application features to reduce redundancies and feature overlaps, leading to cost savings. In such a scenario, organizations would find considerable cost savings in terms of licensing, competing software products, complementary hardware and software costs required to support such applications, and user training. Alternatively, if not, trying to unbundle features from the application may lead to unintended user behavior. If it is found that feature-level usage is dependent on the application, then there is something intrinsic to the application that users respond to in addition to its features. Logically this could be something as simple as the fact that users have been using an application for some time and do not wish to change preferences. It could be that users find certain features to be inherently a part of an application. It could also be that users are captive to certain interfaces and layouts specific to the overall application. Together, it is important to ascertain whether or not users can perceptively separate features from the overarching feature categories and applications.

The second contribution lies in the field of requirements analysis. Understanding perceptions of feature-level usage is likely to increase objectivity in software engineering [Chatzoglou and Soteriou 1999]. Measures of feature usage can help identify value-added versus non-value-added features in applications, allowing developers to concretely focus on delivering the most value for money by accurately selecting application features that best fit the needs of a prospective client. In addition, it can also throw light on how best features can be viably

consolidated for competitive advantage. Because application software consolidation seeks to eliminate feature level overlap among applications, the smallest number of applications to perform the necessary tasks becomes the most cost effective. For example, Microsoft Windows XP is suggestive of feature consolidation, albeit at the operating level, of the power and stability features of the NT platform and the user-friendly features of the 9x platforms. Even to date, feature consolidation is commonly practiced by Microsoft, especially in its service packs that consolidate a set of essential features as a bundled patch for existing operating systems.

Third, drawing on the experiences from five organizations, the paper generates an understanding of how users construct their own perceptions surrounding feature level application usage. Here, the use of Kelly's [1955] personal construct theory is cogent particularly because it offers insight into how individuals construe their perceptions of applications and features leading to their selection and use.

The remainder of the paper is organized as follows: The next section outlines the supporting literature research surrounding the topic of interest. Thereafter, we present our research framework using personal construct theory and the modified repertory grid technique. Next, we discuss the research design and methodology, followed by an examination of the results and a discussion of the findings. Finally, we present the limitations of the study and, subsequently, offer the conclusion.

II. LITERATURE REVIEW

Features are building blocks for any application [Zmud et al. 2004; Griffith 1999; Griffith and Northcraft 1994]. Zmud et al. [2004: 2] goes on to state that "some features reflect the core of the technology by collectively representing the identity or purpose of the technology." Adversely, other features are not key components of an application and their use may be optional [Zmud et al. 2004; Griffith 1999]. There are two differing views regarding the conceptualization of features, the emergent and the appropriation [DeSanctis and Poole 1994; Griffith 1999; Orlikowski 2000]. The emergent focuses on the structures that emerge from the user's repetitive interaction with some technology [Griffith 1999; Orlikowski 2000]. The appropriation conceptualization is defined by the structures inherent in the features of a given technology and observes the appropriation of these structures by its users [DeSanctis and Poole 1994; Orlikowski 2000]. It is probable that both conceptualizations exist "with regard to different individuals in similar situation contexts and with respect to the same individual at different points in time" [Zmud et al. 2004: 5]. Our attempt at understanding the issue utilizes both conceptualizations to provide further insight into feature level usage.

REPERTORY GRID AND PERSONAL CONSTRUCT THEORY

Application users construe and construct their perceptions of an application in alternative and different ways. This philosophy of constructive alternativism argues for the existence of different realities perceived by individuals and underpins Kelly's [1955] personal construct theory (PCT). PCT asserts that individuals create mental constructs to interpret the world around them through interpreting contrasts and similarities [Kelly 1955; Marsden and Littler 2000]. This approach seems particularly useful here given that there is little understanding of how a user construes or mentally models application modularity in terms of features. Kelly's personal construct theory [1955] hypothesizes that individuals attempt to control and predict the events in their lives by creating mental models, hence, the more accurate the mental model, the more predictable the behavior.

The repertory grid technique is the most notable method among the different modes of psychological inquiry generated from personal construct theory [Tan 1999]. Repertory grid creates a focus grid matrix based on cluster analysis of ranked elements within a knowledge domain. The repertory gird technique based on the personal construct theory, once captive to clinical diagnoses, is gaining popularity in business research. For example, PCT was utilized in a

study that found smaller firms in the high technology industry to perceive risk in a fundamentally different way than do their larger counterparts [Sparrow 1999]. In management, a PCT-based repertory grid technique proved a successful method for mapping cognitive constructs of individuals [Dutton et al. 1989; Tan 1999]. For example, Latta and Swagger [1992] used the repertory grid technique as a viable method for modeling knowledge in the development of software interfaces that facilitate human interaction. In this research, the repertory grid technique is used to elicit and link elements and constructs in order to clarify how users perceive applications features.

The repertory grid technique may be decomposed into three parts: elements, constructs, and links. In its most simplistic sense a repertory grid is a table. The elements are the columns and the constructs are the rows. Constructs, defined as polar opposites, reside on both sides of the elements. Kelly [1955] argues that individual personal constructs are bipolar in nature. To clarify, each row represents one construct defined by a pair of polar opposites. It is important to note that it is acceptable for constructs to be rated on a scale allowing respondents to decide their position on an element [Kelly 1955]. The three parts of the repertory grid technique are discussed in further detail as follows.

Elements: An element is the topic or domain of interest under investigation. In this study, elements are software applications currently used within the organizational sample frame. There are two methods of eliciting elements, researcher and participant provided [Hunter 1997; Moyihan 1996]. Researcher-provided element elicitation is employed when the comparison of a standard set of repertory grids among all participants is of primary concern. It is suitable to adopt participant-driven element elicitation when researchers wish to ensure that the elements are relevant and meaningful to a specific group of individuals [Tan and Hunter 2002; Sparrow 1999], particularly because participant-driven elicitation not only offers greater relevance but also mitigates researcher biases. Accordingly, this study uses a Delphi-based iteration technique for elicitation of major elements by participants across five firms.

Constructs: The construct elicitation methods in this study were chosen based on their ability to best predict the software functionality requirements. Similar to element elicitation, there are several different methods of construct elicitation. More precisely, five variations and combinations exist [Beail 1985; Easterby-Smith 1981; Reger 1990; Tan and Hunter 2002]. The construct elicitation methods are researcher-supplied, triading, full-context form, group-construct elicitation, pyramiding, and laddering. In the researcher-supplied method, the researcher supplies predetermined constructs to participants. Previous studies [Latta and Swagger 1992; Phythian and King 1992] have used this method when the primary objective is to measure individuals across different groups. A more popular method, the triading method is a minimum-context method considered to be the classical approach to generating constructs [Tan and Hunter 2002]. In triading, first three elements are randomly selected and participants are asked to state how two of the elements are similar but different from the third. The similarities and differences noted by participants then become constructs. This process of comparing similarities and differences is continued until the researcher is convinced all relevant constructs have been elicited from the participants. It has been suggested by Roger [1990] that seven to ten triads offer an adequate representation of all possible constructs pertinent to most studies. In contrast to a minimum-context method such as triading, the full-context method utilizes all elements simultaneously requiring the participants to sort elements into unique categories and provide several words describing each category. The only theoretical limit to the number of categories is the number of elements. It is important to note that each element may be placed in only one category. The group construct elicitation method [Stewart and Stewart 1981] requires that all participants congregate together to arrive at a group consensus of each construct selected for the study. The group-construct elicitation method is identical to that of triading except it is performed at the group level instead of the individual level. Basically, group construct elicitation takes advantage of open discussions among all participants in its use of the minimum-context method. One of the main benefits of this method is that it allows participants the ability to hear multiple viewpoints on each element [Tan and Hunter 2002]. Finally, laddering is a method that may be used in addition to the methods described previously. Marsden and Littler [2000; pp. 819] describe the laddering method

as “designed to elicit the core values that underpin these bipolar constructs and involves asking why each of the poles are (un)important to them.”

Construct elicitation conducted in this study incorporated a modified version of the full-context form and laddering. The full-context elicitation method gives individual participants, with different wants and needs, the opportunity to create as many constructs as they see fit. The full-context method was used as a brainstorm until the functionality characteristics of each category of software applications were exhausted. The modification made in this study takes the lists created by individuals and presents them to all participant groups for unanimous agreement on features, feature categories, and the applications that best embody them.

In addition to the full-context method, laddering can be used to gain further insight on participants’ understanding of each construct. This process consists of additional questions in the individual interview stage after full-context method is completed. Questions of how and why help obtain meanings behind constructs created by the full-context method [Tan and Hunter 2002]. Previous studies have shown that laddering assists in providing a better understanding of elicited constructs [Phythian and King 1992]. This study used laddering as a technique to clarify constructs elicited by respondent and move them closer to agreement. In this case, laddering was used to both split and merge constructs based on their degree of semantic variance.

Linking: Once constructs and elements are determined, a technique called linking is used to map constructs to elements in three ways [Tan and Hunter 2002]: dichotomizing, ranking, and rating. Dichotomizing is a rather simple procedure that requires researchers to place a check against the element closest to the left pole construct and a cross to the element closest to the right pole construct. This method allows researchers to associate an element with the construct pole that most represents it. A major disadvantage is that the participants must choose only one element that represents a pole of a construct; the method does have the ability to select multiple elements as equals. Another disadvantage is that the selection of only two elements per construct may lead to a skewed distribution [Easterby-Smith 1980]. As a remedy to the skewed distributions of dichotomizing, a ranking method was devised. Ranking requires that participants order the elements as they see fit between the poles of each construct. The problem here is that participants are forced to differentiate among elements on a scale that may not indicate their true differences. The rating method is popular [Hunter 1997; Latta and Swagger 1992] and eliminates the need to select only two elements that best represent each pole of a construct or rank elements based on how well they fit between poles with the ability to assign the same ranking to more than one element [Stewart and Stewart 1981; Tan and Hunter 2002]. Rating requires participants to rate each element based on the given constructs. In this study, the rating method is used to link elements and constructs. The primary reason for selecting this method is that it allows elements to be rated equally on the same construct.

III. METHODOLOGY

The constructs used to populate the repertory grid were obtained by a Delphi-based iterative technique to achieve consensus. The Delphi-based iterative technique was used in the construct elicitation phase to add currency and relevance to the elicitation process. Dalkey and Helmer [1963] state that the objective of the Delphi method is to obtain the most reliable consensus of opinions of a group of participants achieved by a series of questionnaires with controlled opinion feedback. In the current study, participants started with open-ended questionnaires to brainstorm features and feature categories. This process continued until participants identified the same set of features from the preceding round of questionnaires and/or did not ask for further information [Brancheau and Wetherbe 1987]. After initial brainstorming, groups within firm met to finalize the set of features for each category. This iterative modification based on the Delphi method appears to be more conducive to independent thought on the part of the participants and to systematically aid them in brainstorming and elicitation of factors. Delphi procedures are more receptive to independent thought and allow a more gradual transition toward group consensus [Niederman et al. 1991]. The process is designed for the participant to call out all possible issues related to the

problem so that he/she is confident that the question or problem is adequately solved [Dalkey and Helmer 1963]. Niederman et al. [1991] state that the Delphi-based iterative technique is a valuable tool for surfacing new issues and obtaining consensus among a group of experts. In fact, the Delphi-based iterative technique has shown that even when initial participants' views are widely divergent the individual estimates will show a tendency to converge as the process continues. Typically, but not always, the participants are subject-matter experts in a given domain of knowledge.

The participants involved in the study were 29 employees from design and software development departments from five different firms. The five firms in the sample used 59 different software licenses, most of which were not decided by the firm but rather selected and budgeted by each individual department. Licensing was supported by internally allocated funds and driven by choices by managers, developers, or vendors. While the software portfolio was competitive with the general industry, evaluations were focused mainly on overall usability, with little reference to feature overlaps across applications. For example, the instructional design department had recently upgraded a software application for the sake of a single feature (instant messaging) that was already offered by another existing software license. Lacking a rational approach toward determining and consolidating features across the existing application portfolio, it was found that the departments in our sample had software with 35 percent common or redundant features.

Participants for this study included users from different firms: Content Design, Corporate Education and Training, Retail, Software Design, and Transportation. Departments across the firms comprised of the following: one interactive media department with six participants; one instructional design department with seven participants, two application development departments with nine participants, and one database department with seven participants. Because managers promoted our data collection, all on-site departmental members participated. Often more than not, *the developers in the departments also played the role of users*, consuming internally developed solutions as testers or quality controllers.

Table 1. Participant Information

Department	*Interactive Media Development*	*Instructional Design Development*	*Application Development I*	*Application Development II*	*Database Development*
Participants (n)	*6*	*7*	*4*	*5*	*7*
Firm	*Content Design*	*Corporate Education and Training*	*Retail*	*Software Design*	*Transportation*
Firm Size (no. of employees on site)	*>70*	*>150*	*>400*	*>150*	*>200*
Titles (n)	*Digital Media Producer (2)* *Animation (3)* *Project Coordinator (1)*	*Designer/ Developer (3)* *Technical* *E-Training Lead (2)* *E-Learning Specialist (2)*	*API Developer (3)* *Design Architect (1)*	*Developer (2)* *Testing (1)* *Project Management Office (2)*	*Data Quality Analyst (2)* *Database Administrator (1)* *Database Design and Programming (4)*

Participants who informed this study was comprised of developers and users with sufficient expertise in their job functions. The sample included participants whose job functions include graphic design, instructional design, application programming, and database design and development. The diversity of the sample frame is intended to increase generalizability of our

findings across IS professionals. The average age of the participant sample was 35 years with an average of 5.5 years of experience. The participants had an average of 3.3 years of college education and was comprised of eight females (~27 percent).

Each participant essentially served as an IS professional and was classified as subject matter expert (SME) for applications used within their departments. Subject matter experts were well cognizant of the software used in their department, even if they were consistent users of one or more types of application software and casual users of other application software within the department. For example, while some developers used PHP/MySQL and others used a Microsoft .NET framework, both were cognizant of feature availability across domains of expertise.

In Delphi studies, sample size is not a statistical but a quality assurance issue [Duffield 1988]. Because Delphi studies rely on informed experts, it has been suggested that, with a homogenous group of experts, reasonable results can be obtained with small panels of 10 to 15 individuals [Adler and Ziglio 1996; Turoff 1970; Meuleners et al. 2004]. In determining sample size for Delphi-based studies, Duffield [1988] argued that the panel size is a matter of discretion for the researcher. Instead, the sample size in Delphi studies is researcher and situation specific, and more often than not, criterion sampling is chosen to assure quality of responses. The resulting homogeneity allowed for the inclusion of a select body of domain experts as quality informants. In that regard, a sample of 29 domain experts seems to be adequate for this study.

As noted earlier, the methodology adopted is a distinct modification of the repertory grid technique aimed at reducing researcher biases for a more objective assessment, particularly during elicitation. Elements were picked from the application portfolio used by the organizations with careful attention toward making sure that the elements were homogeneous, unambiguous, and representative. Instead of requesting participants to choose elements subjectively, the modification objectively chose the entire application portfolio, thus reducing possibilities of bias and groupthink. Additionally, particular focus was given to make sure that the elicitation of elements used to build the application portfolio was objective with no chances of omissions and blocking.

The key to construct elicitation was to elicit personal constructs though initial brainstorming and then by concatenating semantically redundant themes to arrive at a clarified set of mutually exclusive constructs without ambiguity. Departing from a construct-elicitation technique where the refinement of constructs rests on the researcher, the modification employs a Delphi-based technique for construct elicitation and refinement. Often, during construct elicitation, there is a propensity for researchers to contaminate or bias the elicitation by suggesting constructs. Moreover, because elicited constructs should be comprehensive, it is important to exercise effort in reducing redundancies while making sure that all unique constructs are included. Consequently, a Delphi-based technique was found to be apt for construct clarification and refinement. The Delphi-based technique rests on the respondents and their concerted efforts in iterative refinement of ideas (constructs) so that the final set of constructs are consensual and comprehensive. Instead of relying on the researcher for reliable coding, consolidation, and convergence of constructs, the use of the Delphi technique attests a participant driven process.

The Delphi-based construct elicitation for the repertory grid was iterative. In the first stage, individual construct elicitation (iteration 1) required participants to brainstorm unique application features that they commonly use and assign corresponding categories for every feature. Participants were asked what feature categories, if any, distinguished one or more applications from the rest. Furthermore, participants were requested to identify what specific features these feature categories embodied that differentiated their use (often used for and rarely used for).

The initial brainstorm produced 74 categories and 286 category features. Following the initial brainstorm, participants met as a group virtually (using corporate groupware and private chat rooms) to merge and consolidate categories and respective features. In this second stage of construct elicitation, participants communicated online to consolidate the initial set of categories and features to a clarified list, with few redundant features. Departmental units acted as

independent groups to further merge the features and categories into a final exhaustive list by reiteratively eliminating redundancies and building consensus among departmental participants. In each firm, the final list of features was iteratively developed till consensus was achieved.

Throughout the process, participant anonymity was maintained for open brainstorming and discussion. Group consensus generated a final list of 21 categories spanning 59 applications and 83 features defining the categories. Perceived functionality characteristics of applications were constructs determined by the full context method. After collating all constructs, another virtual meeting consisting of all participants was held to organize individual constructs into a final list. Over subsequent iterations, some constructs remained unchanged, some merged into one or more constructs, and others deleted. The final set of elicited constructs was created only when all participants unanimously converged on the refined set of constructs. On average, it took 3.2 iterations to achieve construct elicitation. Appendix A tabulates the final set of major constructs (features) and elements (applications).

Construct elicitation focused on inter-participant agreement on constructs. Although, the iterative Delphi-based process used in the construct elicitation phase for brainstorming and creating unanimity lessened the extent to which questions and problems needed further clarification, laddering was utilized in instances where there was a degree of ambiguity among participants during construct elicitation. Laddering allowed for an in-depth examination of constructs and was extremely helpful in splitting and merging constructs in the process of iterative convergence of personal constructs. Laddering allowed for a more granular investigation and clarification of constructs to minimize confounds and redundancies during iterative convergence.

In the third stage, participants linked and rated application usage by voting on the degree of use of a particular application for a particular feature. Instead of clustering applications by features as commonly done in repertory grids, our modification asked users to link and rate the degree to which applications were used in terms of specific features. We adopted this particular technique because it offered distinct advantages over repertory grid linking techniques by commonalities between elements by constructs. Primarily, because we are trying to investigate application usage by feature, the difference in usage is that of degree. Because we are measuring perceptions of use, it obviates the need to rank a participant's likelihood of using an application by virtue of its feature set. It would, however, be problematic asking a participant to cluster applications by "often use" or "rarely use" [Hunter 1997; Tan 1999] primarily because users are often captive to a single application although they may have other applications included in their consideration set. They would be likely to use a candidate application and would find it more prudent to rank their likelihood of use by feature. Using the refined list of elicited constructs (nested) within each category, participants rated application usage by each application's respective features on a nine-point scale (linking). Because participants rated their perceived usage of each application per feature, pair-wise comparisons of all features within categories were conducted. The repertory grids are shown in Figure 2.

Finally, participant's rating of application use by features and feature categories allowing for the "linking" elements to constructs. The primary reason for selecting the rating method to link elements to constructs was that it allowed elements to be rated equally on the same construct. However, as shown in the repertory grid, the linking was clustered primarily because specific applications met specific feature requirements. Applications not corresponding to specific features and feature categories were considered irrelevant by participants and not referenced. These applications were treated as "shelfware"- applications adopted but not used because users found them missing required features.

In this study, it was necessary that perceived feature level usage be judged to trace feature similarities across applications. User perceptions of applications sharing similar features under each category were gathered from responses and are denoted in bold and underlined numbers. Following Tan [1999], data from every single feature matrix was averaged for each feature category. The overall perceived alignment between feature categories and applications is indexed for similarity and distinctiveness. For example, participants perceived Fireworks and Flash as

being similar in terms of the features offered within the interactive media category. For feature categories with more than one set of similar applications, the notations are appended with separate color blocks (e.g., the design environment feature category for the instructional design department shows four applications sharing two different sets of features).

Table 2. Advantages and Disadvantages of the Modified Repertory Grid

	Modified Repertory Grid Technique
Advantages	Design Phase: Objective identification of Elements (e.g. from existing logs of application categories)
	Construct Elicitation (Delphi-based: iterative brainstorming: participant led elicitation and convergence (agreement)): Laddering (further clarification of assumptions).
	Multilevel elicitation and convergence (e.g. categories and features)
	Anchoring constructs for single point of reference
	Ranking to assess "likelihood of use" rather than "similarity in use"
	Analysis of nested design to assess hierarchy of use
Disadvantages	Use of Delphi-based convergence a departure for "personal" constructs
	Requirement to achieve consensus may bias construct elicitation and "force" convergence.
	Influence and bias due to power, hierarchy, and groupthink when lacking anonymity

Although the proposed modification to the repertory grid adds value by objective element identification, construct elicitation, and iterative agreement, the weakness of this modification lies in its departure from "personal" constructs. By emphasizing on iterative clarification and agreement, the proposed modification may force participants towards achieving consensus. Furthermore, the anchoring of constructs could possibly bias the process. Again, although due care was taken toward maintaining participant anonymity, issues of achieving agreement and consensus may have been unduly weighed upon by hierarchy, power, and groupthink. Notwithstanding these cautions, we feel that the modification to the repertory grid was useful in our context driven by objectivity and consensus. Table 3 shows the final set of feature categories in bold and their underlying features numbered below.

Table 3. Categories and Feature Definitions (Feature Categories in bold; Application types within parentheses)

Feature Category: Video Editing and Compositing (Graphics Design)	**Feature Category: Interactive Media Creation (Graphics Design)**
Timeline based work	Web publishing
Post-production	Create and edit presentations
Special effects	Non-linear media creation and editing
Multiple image sequence editing	User controlled media creation
Feature Category: Vector Based Graphics Software (Graphics Design)	**Feature Category: Desktop Publishing Software (Graphics Design)**
Edit vector based images	Input data
Create vector based images	Edit text documents

Feature Category: Video Editing and Compositing (Graphics Design)	**Feature Category: Interactive Media Creation (Graphics Design)**
Feature Category: 3D Modeling and Animation (Graphics Design)	Edit spreadsheets
Create 3D objects	Graphics capabilities
3D editing	Simple daily tasks
Animate 3D objects	Feature Category: Raster Based Imaging Software (Graphics Design)
Materials and lighting simulation	Edit raster based images
Computer Aided Design (CAD)	Create raster based images
Visualization tools	**Feature Category: Conversion Software (Graphics Design)**
Feature Category: Viewing Data (Graphics Design)	Convert image based file formats
Viewing images and graphics	Convert video based file formats
Save time in my work	Convert sound based file formats
Streamline production	Complete specialized tasks
Feature Category: Application Model Testing (Application Development)	**Feature Category: Model Compilation (Application Development)**
1. Forward Engineering	1. Open Archetypes
2. Reverse Engineer	2. Execution Engine
3. Mapping Function	3. Stored Instances
4. Mapping Reversibility	4. Call Generation
Feature Category: Development of Console Applications (Application Development)	5. Traverse Capability
1. Database Connectivity	6. Semantic Repository
2. Class Development and Availability	7. Error Detection
3. Web Form Development Interface	**Feature Category: Application Development Environment (Application Development)**
4. Dynamic Testing	1. Predefined Classes and Objects
Feature Category: Code Execution (Application Development)	2. Object and Code Reusability
1. Remote and Local Execution	3. Libraries
2. Object Naming	4. Validation
3. Portability and Executables	**Feature Category: Deployment Versioning (Application Development)**
4. Exception Handling	1. Independent Project Deployment
5. Assembly Verification and Binding	2. Data and Language Migration
Feature Category: Data Control (Database)	3. Security Deployment
1. Rollback Management	4. Version Control
2. Tablespace Management	**Feature Category: Common Language Runtime (Application Development)**

Feature Category: Video Editing and Compositing (Graphics Design)	**Feature Category: Interactive Media Creation (Graphics Design)**
3. Tuning	1. Memory Management
4. Backup and Restoration	2. Thread Management
Feature Category: Database Management (Database)	3. Cross-platform Usability
1. Workload Access and Control	4. Data-Definition
2. Enterprise Management	**Feature Category: Query Development (Database)**
3. Auditing	1. Flashback Version Query
4. Performance Analysis	2. Query Control
Feature Category: Design Environment (Instructional Design)	3. View Control
1. Synchronous Collaboration	**Feature Category: Standards-Based Development (Instructional Design)**
2. Monitoring	1. Automatic Accreditation Updates
3. Survey Development and Administration	2. Competence Analysis
4. Backend Administrative Integration	3. Course Quality Assessment
	Feature Category: Report Development (Instructional Design)
	1. Student Assessment and Scoring
	2. Reporting Options
	3. Content Creation

DATA ANALYSIS

In the linking phase, participants rated their perceived level of use of each feature respective to an application. Data from the repertory grid was analyzed for investigating user perceptions of application level use. Due to the fact that the number of features within applications and applications within categories are different, the design is unbalanced. For example, some application categories have five nested applications each nesting four features within; other application categories may have three applications, each consisting of six features. Since these levels are not identical for applications and features within a particular category, we use a nested design to analyze the unbalanced data. As noted earlier, participants within companies rate every application at the feature level. Since 29 participants were used to rate 21 feature categories, similar ratings had to be accounted for in the model. The list of variables used in the analysis is summarized in Table 4 following.

In this design, categories, applications, and features are fixed for each feature usage score and therefore tests for significance use the Type III sums of squares for fixed effects. Proc mixed within SAS was used to analyze the data. Proc mixed uses a mixed linear model, a technique similar to a generalization of the standard or general linear model, robust in face of some correlation and non-constant variability (SAS Help V9.1). It is held that hierarchically structured data often causes problems with model specification due to clustering effects, and can best be addressed by a mixed linear model [Goldstein 1986]. Within the mixed linear model used for this study, features were nested within applications that were further nested within categories.

Often Used for	Illustrator	Coreldraw				*Rarely Used for*
Vector Based	2	7				Vector Based
	Powerpoint	Flash	Swish	Director	Fireworks	
Interactive Media	6	**4**	6	7	**4**	Interactive Media
	Character3D	Leveleditor	Studio3d	Imagemodel	Viewpoint	
3D Modeling and Animation	**3**	5	**3**	4	6	3D Modeling and Animation
	Thumbsplus	Micro	Fastlook			
Viewing Data	5	**4**	**4**			Viewing Data
	MPEG2	Quicktime	ClearerXL	PDFcreator	Trace	
Conversion	**6**	4	5	6	**6**	Conversion
	Mystical	Paint	Stitcher	Photoshop		
Raster Based	**7**	6	**7**	**7**		Raster Based
	Matchmover	Retimer	Combustion	Edit6	Premiere	Titlemotion
Video Editing and Compositing	**6**	7	**6**	5	5	**6**	Video Editing and Compositing

Often Used for	Visio EA	VP-UML	Charter	Diablo	Borland	Oracle BPEL (Remote)	Visual Studio .NET	ASP Express	Websphere	Microsoft SourceSafe	J2EE	Jrun	VBA	*Rarely Used for*
Application Model Testing	**7**	**7**	4	1	1	1	1	1	1	1	1	1	1	Application Model Testing
Model Compilation	**6**	1	1	5	3	4	**3**	1	1	1	1	1	1	Model Compilation
Deployment Versioning	1	1	1	1	1	1	**7**	4	**5**	**4**	**5**	5	1	Deployment Versioning
Common Language Runtime	1	1	1	1	1	1	1	1	1	7	**3**	1	**3**	Common Language Runtime
Development Environment	1	1	1	1	1	1	1	1	1	5	**7**	1	**5**	Development Environment

Often Used for	ID Expert	CentraONE	Lotus LearningSpace	Blackboard	elluminate Vclass	Mentargy LearnLinc	Web Surveyor	*Rarely Used for*
Standards-Based Development	3	**6**	**6**	1	1	1	1	Standards-Based Development
Report Development	1	**5**	7	**4**	1	1	1	Report Development
Design Environment	1	**6**	**4**	1	**2**	**2**	5	Design Environment

Often Used for	Oracle	MySQL	SQL Server	Apex SQL	J2EE (ODBC)	Oracle LogMiner	*Rarely Used for*
Data Control	**3**	4	**7**	1	1	6	Data Control
Database Management	**5**	1	**6**	2	5	1	Database Management
Query Development	**6**	1	**5**	1	1	4	Query Development

Figure 2. Repertory Grid of Applications (Elements) and Constructs (Feature Categories)

Table 3 following contains the results of the mixed linear model analysis. The mixed model uses the F test to assess the significance of the predictors in the model. The three predictor variables: (1) features nested within applications that are nested within categories; (2) applications nested within categories; and (3) standalone categories, were found to significant at an alpha of .05. These results show that feature categories, applications, and features are important in assessing usage. Moreover, it is interesting to note that users perceive feature categories and applications to be relatively more important than features.

Table 4. Variables Used in the Study

Variable	Scale	Levels	Type	Interpretation
Category	Nominal	21	Predictor	This variable translates to the category of applications the feature under measurement falls in
Application	Nominal	Varying	Predictor	This variable equates to the application the feature under examination is a part of
Feature	Nominal	Varying	Predictor	This variable indicates the current feature under measurement
Person	Nominal	29	Repeated measure	This variable corresponds to the participant responding to his feature level usage
Usage	Ordinal	7	Response	This variable is the feature level rating scores of the participants

Table 5. Significance of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
Feature Categories	21	84	18.99	<.0001
Applications	59	236	9.01	<.0001
Features	83	332	1.41	<.05

IV. RESULTS AND DISCUSSION

Altogether, the results provide some interesting findings. First, the repertory grid reiterates the concerns organizations have regarding redundant applications. Nearly all participants confirm the adoption of multiple applications with redundant features. The issue is particularly acute in the context of deployment versioning. Data from application developers suggested four applications sharing the same set of features. On the other hand, the database group seemed relatively more prudent in their choice of applications.

Differences in work outcomes may explain differences between these groups. Graphics designers seem to be very discerning yet subjective with their choice of specific application features (revealed by their large choice of feature categories) relative to database designers who look at much more objective, data-driven characteristics in their application features, traits also shared by

the instructional design group. Application developers, on the other hand, indicated how discriminating they were with particular features, preferring a set of programming software applications for ad hoc feature use rather than "churn all deliverables through a single mill."

Second, results show that participants regard both applications and feature categories as important dimensions guiding their choice of software. This implies the problem users and developers may undergo if asked to cognitively delineate features or feature categories from an application. For example, features such as "control panel" and "regedit" are often associated with Microsoft Windows operating system. Mentally, it may be difficult for users to disassociate these features as being application, or in this case- operating system, independent. If so, the finding implies two important facts. First, users perceive an application to be more than a sum of its features. Such a position reduces feature independence, offering vendors to capitalize on user dependence on interfaces or otherwise by bundling features to a popular application. If user perceptions are largely application-centric, it becomes difficult for firms to rationalize feature-level usage. Second, it supports vendor strategies of cognitive "lock-in," where a particular feature is automatically associated with a specific application or category. For example, a Microsoft Windows operating system user who likes the feature of "right-clicking" a mouse to open up a set of functions is likely to feel distanced by operating systems (e.g. pre OS X Macs) that do not support that feature. Interestingly though, the data provides a slight but interesting departure. While the perceived importance of feature categories and applications received significant support ($Pr>F < 0.0001$), the importance of specific features only received marginal support ($Pr>F < 0.05$). Such a finding could be suggestive of the fact that users may be gradually perceiving features as being independent of an application vis-à-vis their perceived dependence afforded to feature categories in applications.

The growing standardization of features under feature categories offers a case in point. For example, in systems development and deployment, the feature category of deployment versioning has evolved to encompass a standard set of features (functions). Users are more interested in knowing whether an application includes the "deployment versioning" feature category than what features comprise the feature category. There may be a growing assumption that the existence of a feature category toolset will most likely encompass the necessary features. Moreover, because a large part of an organizational portfolio consists of application software from reputed vendors, developers and users are well aware of the toolset offered by different feature categories. While the results show that developers and users significantly value applications, it also points out that developers' and users' application choices may be driven by availability of feature categories (note the large shift in F-value for feature categories over applications). It could be argued that an attempt to reduce the set of applications used in current organizations could result in a negative perception among users; in the same vein, it could likewise be argued that the same users' choice of applications are triggered by the availability of feature categories that they require and deem fit for use. This element of rationality may be utilized by organizations to trigger a reduction in its application portfolio based on feature level overlap. Vendors may capitalize on the same rational streak by marketing the overall feature category for their applications rather than individual features. For example, a programming platform vendor may find it more useful to stress on the ease of use of the overall development environment rather than by explaining each feature. Both organizations and vendors may have to recondition themselves by realizing that software choices among developers and users are perhaps notionally linked to reducing information overload.

Of the two differing views regarding the conceptualization of features—emergent and appropriation [DeSanctis and Poole 1994; Griffith 1999; Orlikowski 2000]— we find support for both. In accordance with the appropriation conceptualization, users operate within a bounded structure of applications, constrained by their inherent features. During the laddering stage, both developers and users noted that they used many of the features as designated by the specific application. However, developers and users also noted that they often used features in a way not only based solely on vendor specifications but also in ways that allowed them to best complete their work, a condition matching emergent conceptualization. It is plausible that participants utilize

features in both existing and innovative ways by finding substitute and alternative modes of operation to overcome and extend their learning curve to accomplish the task on hand.

The concepts of coupling and cohesion add considerable explanatory value. Coupling relates to how closely tied a certain feature (as a program module) is to other features or other environmental (e.g. application, platform) modules in general. Features are coupled into applications, i.e., feature level modules are dependent on application level modules for functioning. Cohesion, on the other hand, refers to the overall functionality and reusability of the features within an application. In general, a loosely coupled module would have higher cohesion because they can be reused for optimal return. How strong features are (or perceived as) coupled in an application may well be the clue behind understanding whether users can perceptively decouple features. An interesting and common example is that of Web applications. A typical shopping cart application embeds a login feature and credit card verification feature. Although these features are easily separable program modules (i.e. easy to decouple and thus more cohesive), the tight and complex linkages (coupling) of these features within the application often makes it perceptively difficult to decouple them. This strategy is perhaps the most fruitful for vendors. Here, vendors, in reality, can create loosely coupled modules that are easily callable (reusable) by other application modules to maintain a high level of cohesion. If the cohesion is high enough, users are most likely to perceive these features as cognitively inseparable from an application, thus increasing “lock-in” effects and user loyalty toward applications.

Similarly, parallel arguments may also explain user perceptions of feature use. In order to maintain a relatively objective view of features within applications, this study is limited in its primary assumption of homogeneity within features. However, in reality, feature quality is an important aspect. Consider how the regression feature considerably varies across SAS versus SPSS. Vendors that can enhance feature quality within an application are more likely to create better lock-in effects for users, thus making the application an extension of the feature. Furthermore, a competing argument can also be drawn from the overall application and platform dependence of features. How open is the hardware and operating system environment for supporting features? User perceptions of feature use are often a function of operating and hardware platforms. For example, if an application is unsupported by a particular platform, users’ perceptions toward feature-level independence that allows them to use the features independent of the application are more likely to be positive and significant.

V. CONCLUSION

Our research finds that users perceptively differentiate, albeit marginally, individual features from overall applications but not to the same extent as they do feature categories. Among users (including developers), perceptions of application-level use are more significant than feature-level use, the importance of applications is still perceived to be greater than its specific features. Users seem to view applications as personally “constructed convenient fictions for describing and discussing particular constellations of features” [Griffith and Northcraft 1994:283]. Individual features seem to be subsumed by the application itself and users do not place features as being more important than applications. Instead, users and developers perceive that features are captive to particular applications—it is difficult separating them. However, it is interesting to note that users perceive feature category use to more important than individual features. Feature categories remain core, suggestive of being critical to defining the choice of an application while individual feature specifications become optional, intended to enhance the application beyond its core features [Griffith 1999], but not guide the choice of application.

Inferentially, one can argue that user perceptions of use tend be married more to an application environment than to a particular feature. If a feature were to be separated from that particular (application) environment, users may feel distanced in the use of that particular feature. The layout or interface of the application environment can perhaps impact usage more than application features. For example, although statistical analysis features (e.g. ANOVA, cluster analysis) are available across multiple applications (SAS Base, Minitab, SPSS…), users tend to

use different applications prompted by their perception of the core feature categories (e.g., Difference of Means, Multivariate Analysis) in the belief that availability and ease-of-use of core categories are more important that individual features. This view could be strengthened by future research using a larger sample to investigate whether the marginal support received by features still held true.

In many cases, unbundling of an application from its features is inconceivable. Here, feature overlaps may commonly be disregarded for a higher level of comfort with particular application environments. From an adaptive theoretical standpoint, which posits how the interplay of user and technology leads to different mutable realities [DeSanctis and Poole 1994], it could also be argued that the interplay of users with technology (applications and features) allows structures (equilibrium) to merge over time. The equilibrium is simultaneously rational and perceptual. It is rational because users and developers require the availability of particular feature categories as a toolbox of pertinent features. It is perceptual because the overall application is perceived to be more important than individual functionalities or features. Although the equilibrium is fragile, user action at our cross-sectional point of reference shows emergence of a structure that favors feature categories and applications over individual features.

User cognition also provides a certain degree of clarification. At feature level, Griffith [1999] notes, technology can be concrete or abstract: concrete features are objective and can be directly specified and distinguished; abstract features are much more subjective and difficult to describe. While triggers from concrete features (e.g. specific functions) are easier to capture and analyze, abstract features are more complex to define (e.g. feeling about an application environment, the way features are combined). This study found user and developer inclination more towards abstract, rather than concrete. The choice of abstract nature of feature categories as driving application choices vis-à-vis the concrete nature of individual features refers to a sense of complexity surrounding user cognition and choices. Perhaps future attempts at capturing the direct and indirect effects of concrete and abstract features may offer a richer and different perspective on users' and developers' perception and sense making.

Another future direction for this study can stem from investigating moderating influences such as organizational culture, management style, and organizational practices on feature usage. The culture and volatility of policies, practices, and markets often prompt different user and developer behavior. Particular industries are more susceptible to software innovations and upgrades than others. Certain industries in fast-paced markets prompt user and developer demand for newer software and upgrades to meet changing needs. In our particular study, the speed at which these organizations phased software in and out was fast and the need for proficiency in software applications thus high. For companies and industries where software turnover is high, it sometimes becomes difficult keeping abreast of all feature level innovations, forcing in a new mindset that examines features more at a category level than at specific technical levels. Also, as one user suggested, "the vendors keep renaming the same features….one time or the other you will end up losing track." In such an instance, developers and users would most likely fall back on particular feature categories and specific applications to reduce information overload and complexity surrounding relearning and use. Perhaps the equation would shift in companies and industries with a less volatile application portfolio. One could even contend that the shift in feature dependence could also be a of developer and user proficiencies.

Again, there could also be considerable differences in individual thought processes between those persons using highly technical and non-technical applications, such as Word Processing versus Computer Aided Design (CAD). This study does not look into these issues. Further research should investigate software selection practices based on complexity of the software application and also between industries.

In summary, it is certain that organizations will keep on building their application portfolio and applications will keep growing in complexity, adding more and more features to its current portfolio. These features, both core and tangential, can be objective or abstract. We can glimpse into how user interaction at an application and feature level allows a socio-technical system to

emerge. Organizations and vendors can better manage such socio-technical systems when they have a better acumen toward what drives user choices and whether feature independence will evolve as a credible trend. It is important to know if users are rational enough to driven by objective feature-based criteria or are captive to particular application environments. Our study finds partial support for both. While developers and users deem feature categories important, they are yet to perceive a software application merely as a sum of its parts (features). Instead, a gestalt perspective, suggesting that both applications and feature categories are perceptively greater than just a sum of their underlying features, seems to be in vogue. Findings from this study attest to such a gestalt perspective.

REFERENCES

Adler, M., and E. Ziglio. (1996). *Gazing into the Oracle: The Delphi Method and Its Application to Social Policy and Public Health*, Jessica Kingsley Publishers, London.

Beail, N. (1985). "An Introduction to Repertory Grid Technique," In N. Beail (Ed.), *Repertory Grid Technique and Personal Constructs*. London: Croom Helm, pp. 1-24.

Brancheau, J. and J. Wetherbe. (1987). "Issues In Information Systems Management," *MIS Quarterly*, 11(1), 23-45.

Chatzoglou, P. and A. Soteriou. (1999). "A DEA Framework to Assess the Efficiency of the Software Requirements Capture and Analysis Process," *Decision Sciences*, 30 (1), pp. 503-532.

Compeau, D. and C. Higgins. (1995). "Computer Self-Efficacy: Development of a Measure and Initial Test," *MIS Quarterly*, 19 (2), pp. 189-211.

Dalkey, N. C. and O. Helmer. (1963). "An Experimental Application of the Delphi Method to the User of Experts," *Management Science*, 9, 3, pp. 458-67

Davis, F. (1989). "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, 13 (3), pp. 319-340.

DeSanctis, G. and M. Poole. (1994). "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory," Organization Science, 5, pp. 121-147.

Duffield, C. (1988). "The Delphi Technique," *The Australian Journal of Advanced Nursing* Vol. 6, pp.41-5.

Easterby-Smith, M. (1981). "The Design, Analysis and Interpretation of Repertory Grids," in Shaw, Ed. *Recent Advances in Personal Construct Theory*. Academic Press: New York, NY.

Goldstein, H (1986). "Multilevel Mixed Linear Model Analysis Using Iterative Generalized Least Squares," *Biometrika*, Vol. 73, No. 1 (April), pp. 43-56.

Griffith, T. (1999). "Technology Features as Triggers for Sensemaking," *Academy of Management Review*, 24, pp. 472-488.

Griffith, T. and G. Northcraft. (1994). "Distinguishing between the Forest and the Trees: Media, Features, and Methodology in Electronic Communication Research," *Organization Science*, 5, pp. 272-285.

Hunter, M. (1997). "The Use of Repgrids to Gather Interview Data about Information Systems Analysts," *Information Systems Journal*, 7, pp. 67-81.

Jasperson, J., P. Carter, and R. Zmud. (2004). "A Comprehensive Conceptualization of the Post-Adoptive Behaviors Associated with IT-Enabled Work Systems," *MIS Quarterly*, Vol. 29: 3, pp. 525-557.

Kelly, G. (1955). *The Psychology of Personal Constructs*, Vol. l. Norton: New York.

Latta, G. and K. Swagger. (1992). "Validation of the Repertory Grid for Use in Modeling Knowledge," *Journal of the American Society for Information Scienc*e, 43 (2), pp. 115-129.

Marsden, D. and D. Littler. (2000). "Repertory Grid Technique: An Interpretive Research Framework," *European Journal of Marketing*, 34 (7), pp. 816.

Meuleners, L. B., R. Cercarelli, and A. Lee. (2004). **"**Issues Affecting Heavy Vehicle Drivers in Western Australia: A Delphi Study," *Road and Transport Research*, December, pp. 1-8.

Moore, G. and I. Benbasat. (1991). "Development of an Instrument to Measure Perceptions of Adopting an Information Technology Innovation," *Information Systems Research*, 2 (3), pp. 192-222.

Moyihan, T. (1996). "An Inventory of Personal Constructs for Information Systems Project Risk Researchers," *Journal of Information Technology*, 11, pp. 359-371.

Niederman, F., J. C. Brancheau, and J. C. Wetherbe. (1991). "Information Systems Management Issues for the 1990s," *MIS Quarterly*, 15(4), 475 - 500

Orlikowski, W. (2000). "Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations," *Organization Science*, 11 (4), p.404-428.

Phythian, G. and M. King. (1992). "Developing an Expert System for Tender Enquiry Evaluation: A Case Study," *European Journal of Operational Research*, 56 (1), pp. 15-29.

Reger, R. (1990). "The Repertory Grid Technique for Eliciting the Content and Structure of Cognitive Constructive Systems," In *Mapping Strategic Thought* (Huff, Ed.), pp. 301-309. Wiley: Chichester.

SAS Help V9.1 for Windows. Cary, NC: SAS Corporation.

Sparrow, J. (1999). "Using Qualitative Research to Establish SME Support Needs," *Qualitative Market Research*, Bradford 2 (2) pp. 121.

Stewart, V. and A. Stewart. (1981). *A Business Application of Repertory Grid*. McGraw Hill: London.

Tan, C. (1999). "Exploring Business-IT Alignment Using the Repertory Grid," Proceedings of the 10th Australasian Conference on Information Systems, pp. 931-942.

Tan, C. and M. Hunter. (2002). "The Repertory Grid Technique: A Method for the Study of Cognition in Information Systems," *MIS Quarterly*, 26 (1), pp. 39-57.

Thompson et al., (1991). "Personal Computing: Toward a Conceptual Model of Utilization," *MIS Quarterly*, 15 (1), pp. 125-143.

Turoff, M. (1970). "The Design of a Policy Delphi," *Journal of Technological Forecasting and Social Change,* Vol. 2, pp.149-172.

APPENDIX A: APPLICATION USAGE ON A PER FEATURE BASIS

Feature	Application(s) (Rep Grid)	Category
Web publishing	Flash, Fireworks	Interactive
Creating and editing presentations	PowerPoint, Fireworks	Interactive

Feature	Application(s) (Rep Grid)	Category
Non-linear time-based media creation & editing	Flash, Fireworks	Interactive
User-controlled media creation	Swish, Fireworks, Director	Interactive
Editing vector-based images	Illustrator	Vector-based
Creating vector-based images	CorelDraw	Vector-based
Input data	Excel	Desktop Publishing
Edit content	Word, PageMaker	Desktop Publishing
Edit spreadsheets	Excel	Desktop Publishing
Graphics capabilities	Word, PageMaker, WordPerfect	Desktop Publishing
Simple day-to-day tasks	Word, PageMaker	Desktop Publishing
3D editing	Character 3D	3D Modeling
Animating 3D objects	Image Modeler	3D Modeling
Materials and lighting simulation	Level Editor	3D Modeling
Computer Aided Design (CAD)	ViewPoint	3D Modeling
Visualization tools	Viewpoint	3D Modeling
Creating 3D objects	Character 3D, 3D Studio, ImageModeler	3D Modeling
Viewing images and graphics	Thumbsplus, Fastlook	Viewing Data
Save time in my work	Micro	Viewing Data
Streamline production	Micro	Viewing Data
Convert image based file formats	PDF Creator	Data Conversion
Convert video based file formats	MPEG2, QuickTime	Data Conversion
Convert sound based file formats	QuickTime, CleanerXL	Data Conversion
Complete specialized tasks	Trace	Data Conversion
Creating raster-based images	Photoshop, Paint	Raster-based
Editing raster-based images	Mystical, Stitcher	Raster-based
Timeline-based work	Matchmover, Titlemotion, Edit6	Video Editing
Post-production	Combustion, Edit6, Primiere	Video Editing
Special effects	Combustion	Video Editing
Multiple image sequence editing	Retimer, Combustion, Edit6	Video Editing
Forward Engineering	Visio, VP-UML	Application Model Testing
Reverse Engineer	Visio, VP-UML	Application Model Testing
Mapping Function	Charter, Visio, VP-UML	Application Model Testing
Mapping Reversibility	Charter, Visio	Application Model Testing
Open Archetypes	Borland, Visio EA, Diablo	Model Compilation
Execution Engine	Borland, Visio EA, Visual Studio, Diablo	Model Compilation

Feature	Application(s) (Rep Grid)	Category
Stored Instances	Visual Studio, Borland	Model Compilation
Call Generation	Visual Studio, Borland, Diablo	Model Compilation
Traverse Capability	Diablo, Visual Studio	Model Compilation
Semantic Repository	Oracle BPEL (Remote), Visual Studio	Model Compilation
Error Detection	Visual Studio, Borland, Diablo	Model Compilation
Independent Project Deployment	ASP Express, WebMatrix, Visual Studio .NET, JRun	Deployment Versioning
Data and Language Migration	.NET SDK, Visual Studio .NET, Websphere	Deployment Versioning
Security Deployment	Microsoft SourceSafe, TeamSource DSP, J2EE	Deployment Versioning
Version Control	Microsoft SourceSafe, Team Source DSP	Deployment Versioning
Assembly Verification and Binding	Visual Studio .NET, Websphere	Deployment Versioning
Memory Management	Visual Studio .NET, J2EE	Common Language Runtime
Thread Management	Visual Studio .NET, J2EE	Common Language Runtime
Cross-platform Usability	Visual Studio .NET, J2EE	Common Language Runtime
Data-Definition	Visual Studio .NET, J2EE	Common Language Runtime
Predefined Classes and Objects	Visual Studio .NET, J2EE, VBA	Development Environment
Object and Code Reusability	Visual Studio .NET, J2EE, VBA	Development Environment
Libraries	Visual Studio .NET, J2EE	Development Environment
Validation	Visual Studio .NET, J2EE	Development Environment
Workload Access and Control	SQL Server, Oracle, J3EE (JDBC)	Database Management
Enterprise Management	SQL Server, Oracle	Database Management
Auditing	SQL Server, Apex SQL	Database Management
Performance Analysis	SQL Server, Oracle	Database Management
Flashback Version Query	Oracle LogMiner	Query Development
Query Control	SQL Server, Oracle	Query Development
View Control	SQL Server	Query Development
Automatic Accreditation Updates	ID Expert, CentraONE	Standards-Based Development
Competence Analysis	CentraONE	Standards-Based Development
Course Quality Assessment	Lotus LearningSpace	Standards-Based Development

Feature	Application(s) (Rep Grid)	Category
Student Assessment and Scoring	Lotus LearningSpace, CentraOne	Report Development
Reporting Options	Lotus LearningSpace	Report Development
Content Creation	Blackboard, CentraOne, Lotus LearningSpace	Report Development
Synchronous Collaboration	Lotus LearningSpace, elluminate Vclass	Design Environment
Monitoring	Mentargy LearnLinc	Design Environment
Survey Development and Administration	Web Surveyor, Blackboard, CentraOne	Design Environment
Backend Administrative Integration	Lotus LearningSpace	Design Environment

ABOUT THE AUTHORS

Michael Harrison is a doctoral student in the Heinz School of Public Policy and Management at Carnegie Mellon University. Michael has several years of experience as a consultant for Deloitte and Touche and as an entrepreneur. He is interested in software development, use, and reengineering and is currently involved in the computational analysis of organizational systems.

Pratim Datta is an Assistant Professor of Information Systems in the Department of Management and Information Systems at Kent State University. Pratim has a PhD and MS from Louisiana State University. Pratim has been involved in security, design, and reengineering of organizational information systems. Pratim is also interested investigating the economics and psychology of user behavior in a variety of technology settings. His research has been published and presented in a variety of journals and conferences, notably the *Journal of the Association of Information Systems*, *IEEE Transactions*, *Communications of the Association of Information Systems*, *Communications of the ACM*, *International Conference on Information Systems*, and the *European Conference on Information Systems*.

Communications of the Association for Information Systems

ISSN: 1529-3181