JSS 7268
DISK / 26/9/01

ARTICLE IN PRESS

No. of pages: 17
DTD 4.2.0/ SPS

ELSEVIER

The Journal of Systems and Software xxx (2001) xxx–xxx

The Journal of Systems and Software

www.elsevier.com/locate/jss

An estimation of the decision models of senior IS managers when evaluating the external quality of organizational software

Bonnie Brinton Anderson, Akhilesh Bajaj *, Wilpen Gorr

The H. John Heinz III School of Public Policy and Management, 2105 C Hamburg Hall, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Received 21 September 2000; received in revised form 23 January 2001; accepted 22 March 2001

Abstract

The increased usage of software in large corporations, coupled with the explosion in software availability has made it important to evaluate software quality (SQ) from the point of view of its consumers. In this study, we focus on the external quality (quality from the point of view of the consumer) of off-the-shelf software used in large corporations. We refine external software quality into four factors. Next, we utilize conjoint analysis (CA) to study the relative values of these factors in the decision models of senior IS managers, when evaluating software for use by their organization. Our results indicate that software used in large corporations today has evolved, so that, contrary to earlier studies that indicate *learnability* and *features* as important, it is now the *reliability* of the software that is the primary factor in IS managers' decision models. The findings have implications for IS theory, and provide guidelines for resource allocation for software developers, IS managers, researchers in the area of software reliability and designers of IS curricula. © 2001 Published by Elsevier Science Inc.

Keywords: External software quality; Conjoint analysis; Information system managers; Organizational software; Software reliability; Software learnability; Software response time; Software features

1. Introduction

During the past 15 years, there has been an explosive growth in computer technology applications, and the software industry has been growing by orders of magnitude (Cusumano and Kemerer, 1990). Competition has also intensified with a multifold increase in the number of firms producing software (Kekre et al., 1995). With increased competition, it has become more important for software firms to understand what factors, that describe software, are important in the minds of their potential customers. Depending on the software, [1] the consumers of the software can be either end-users or the IS staff in an organization.

When evaluating software, the *quality* of the software is widely accepted to be one of the most important factors in the decision models of potential consumers, regardless of the particular application domain (Fenton, 1991; Florac, 1992). A commonly used (Barbacci et al., 1995, 1997) definition of software quality (SQ) is that *it is the degree to which software possesses a desired combination of attributes* (IEEE, 1992).

In the past, SQ has often been classified into internal SQ, which is the quality from the point of view of the creators (programmers) of the software; and external SQ, which is the quality from the point of view of the consumers of the software. Examples of internal SQ are McCabe's Cyclomatic method (McCabe, 1976) and measures derived from Halstead's software science (Halstead, 1977), while examples of external SQ are reliability and usability from the point of view of the users. In this work, we focus exclusively on external SQ.

There is considerable previous work on SQ. However, to the best of our knowledge, there has been only one study that has looked at the relative weights of some of the factors that constitute external SQ (Kekre et al.,

* Corresponding author. Tel.: +412-268-4271; fax: +412-630-8840.
E-mail addresses: bbrinton@andrew.cmu.edu (B. Brinton Anderson), akhilesh@andrew.cmu.edu (A. Bajaj), wg0g@andrew.cmu.edu (W. Gorr).

[1] The software used in an organization can be split into several levels, starting from the operating system at the bottom, moving up to application systems like database management systems, moving up to end-user applications such as database form applications. Each level's quality depends on the levels below it. In this study, we define software as all the software that all members of the organization would use. Thus, IS staff may interact with the operating system and the next higher level, while end-users may react only with the highest levels. Ultimately, the goal of an organizational IS is, of course, to deliver end-user software.

0164-1212/01/$ - see front matter © 2001 Published by Elsevier Science Inc.
PII: S0164-1212(01)00114-5

Table 1
List of external SQ factors listed from previous works

Factor name	Description	Some references
Reliability	What confidence can be placed in what it does/does it behave as intended?	Barbacci et al. (1995), Bays (1995), Keller et al. (1990, 1995)
Survivability	How will it perform under adverse conditions?	Keller et al. (1990)
Usability	How easy is it to use?	Keller et al. (1990, 1995)
Correctness/capability	How well does it support features that are needed by the user?	Bays (1995); Keller et al. (1990, 1995)
Maintainability	How easy is it to repair/upgrade?	Barbacci et al. (1995), Keller et al. (1990, 1995)
Interoperability	How easy is it to interface with other systems?	Keller et al. (1990)
Installability	How easy is it to install?	Kekre et al. (1995)
Performance	How quickly does it respond to users?	Barbacci et al. (1995), Bays (1995), Kekre et al. (1995)
Documentation	What kind of user manuals and online help are available?	Kekre et al. (1995)

Table 2
Definitions of factors identified in (Bajaj, 2000) as affecting the evaluation models of senior IS managers regarding computing architectures for their organizations

Factor	Definition
Software quality	The quality of software associated with the architecture. This can include response time to end-users, quality of user interface, and features provided by the software
Centralization v/s distributed nature	A centralized architecture means that software resides in a centralized location, and most of the hardware investment is also centralized
Costs	The costs of a architecture include the costs of acquisition of hardware, software, the costs of maintenance of hardware, of controlling different versions of the software and the costs of personnel trained in maintaining the hardware and software
Acceptance of the architecture	This factor represents the degree to which a particular architecture has been accepted by IS magazines, the media, model organizations, and software and hardware vendors
Backward compatibility of the architecture	This factor models the degree to which a architecture will cause changes in the organization. Changes include: converting old data to be read by the new architecture, retraining users to use and IS personnel to maintain the software and hardware

1995). There is thus still a need to learn more about the different factors that constitute external SQ, and the relative weights of these factors. The primary contribution of this work is the identification and measurement of the relative weights of the factors that senior IS managers consider, when they evaluate software for their organization.

The question we are addressing here is of interest not only to IS theory, but also to IS managers in industry. The answers to this question will (a) give senior IS managers insight into the decision models of their peers; (b) identify the drivers of competitive advantage for software producers and (c) identify areas of training for future IS managers in the IS curricula in our universities.

2. Previous work on external software quality

The need to decompose internal and external SQ into more refined factors was recognized early (Boehm et al., 1978). Since then, there has been considerable work on identifying the factors that constitute external SQ. A general consensus seems to be that external SQ is dependent upon the intended use of the system (Barbacci et al., 1997; Boehm et al., 1978; Cardenas-Garcia, 1991; IEEE, 1992; Keller et al., 1990). Thus, external SQ can be decomposed differently depending on the set of users and the intended use of the software. *The current study focuses on the external SQ of off-the-shelf software that is used by employees of large organizations in the USA.*

An extensive list of factors that relate to the external SQ of this type of software can be obtained from previous literature. Table 1 contains a list of factors that have been identified by us after reviewing past works.

The factors listed in Table 1 represent different aspects of SQ from the point of view of the users. They are reasonably non-overlapping, and it is conceivable that all of these factors may be important. To the best of our knowledge, only one study (Kekre et al., 1995) has looked at the relative weights of some of these factors. Based on interviews with focus groups, Kekre et al. (1995) listed seven factors as being important for eight different IBM software products used by corporations. These seven factors are all included in Table 1. Of these factors, *capability* was found to correlate most highly with overall customer satisfaction, which was self-reported. *Usability* was found to be the most important factor for end-users of the software, while *capability* and *reliability* were more important for systems programmers.

In a study on the identification and relative weights of factors that drive senior IS managers' evaluation of

Table 3
Factors describing external SQ identified in the current study and how they map to those used in earlier studies

Reference	Features	Learnability	Reliability	Response time
Barbacci et al. (1997)	Non-occurrence of the improper alteration of information	–	Probability that the system will continuously provide outputs over a specified amount of time or the ability to keep operating over time, MTTF, readiness for usage	Time elapsed between the arrival of an input and its corresponding output to the environment
Boehm et al. (1978)	All of its parts are present and fully developed	Convenient and practicable to use, contains uniform notation, and terminology, without any excessive information	Can it be expected to perform its intended functions satisfactorily?	Fulfills its purpose without a waste of resources
Cardenas-Garcia (1991)	Features implemented	Time to learn, retention over time, user satisfaction, overall design quality	Probability of failure-free operation, failure rates for a specified environment that is deemed allowable by the user	Performance with respect to individual users, distribution of arrival times, workloads and service time
Christie (1994)	–	Ease of learning, ease of use, clarity of presentation and documentation, quality of on-line help	–	Space and execution time efficiency
Fenton (1991)	–	User-friendliness or the probability that the operator of the system will not experience a user interface problem during a given period of operation	Specific measures	
Florac (1992)	Completeness, correctness	Usability	Reliability–IEEE standard	Efficiency
IEEE (1992)	Existence of certain properties and functions that satisfy stated or implied needs of users	Amount of user effort required to understand software; the degree to which user effort required to understand software is minimized; the effort needed for use and the individual assessment of such use by users	Capability of software to maintain its level of performance under stated conditions for a stated period of time; survivability; the degree to which software can detect and prevent information loss, illegal use, and system resource destruction	Relationship of the level of performance to the amount of resources used under stated conditionstime and resource
Kekre et al. (1995)	Capability: customer satisfaction with the functionality in terms of key feature offered	Usability: initial effort to learn a software product and the recurring effort required to use the product	Reliability assesses the extent of disruption by failures	Performance
Keller et al. (1990)	How well does it conform to the requirements?	How easy is it to use?	What confidence can be placed in what it does? How well will it perform under adverse conditions? How secure is it?	How well does it utilize a resource?
QAI (1989)	Functional requirements	Ease of use, adequacy of documentation	System reliability, accuracy of outputs, data security	Response time, timeliness of outputs

computing architectures [2] for their organizations, Bajaj (2000) used semi-structured interviews to identify five factors that are considered by senior IS managers when evaluating architectures (see Table 2). The findings indicate that the external quality of software is the most important factor in the evaluation models of senior IS managers. In the current work, we refine external SQ into component factors, and determine the relative weights of these factors in senior IS managers' evaluation models. Our findings here will provide more insight into which factors in external SQ are the drivers of better perception of SQ, and hence to the perception of computing architectures by senior IS managers.

[2] In (Bajaj, 2000), a computing architecture is defined as a new computing infrastructure that significantly affects the purchasing and maintenance of hardware and software in an organization. Examples include a main-frame architecture with dumb terminals, a client/server architecture and a network computers architecture.

Table 4
Definitions of external software quality factors used in this study

Factor	Definition
Features	Features of software associated with an architecture support what end-users do with the software. E.g., there is a certain set of core tasks that end-users do with financial data, and software for managing this would have features that support these core tasks. If the software has *sufficient* features then all core tasks will be supported. If the software has *less* features then some core tasks will not be supported
Learnability	Learnability of software associated with an architecture measures how easy it is for end-users to learn to use the software initially and to remember how to use the software subsequently. E.g., if the software for managing financial data has *high* learnability, then it will be easier for end-users to learn to use it and to remember how to use it. If the software has *low* learnability then it will be less easy for end-users to learn to use it and to remember how to use it
Reliability	Reliability of software associated with an architecture is the extent to which it functions as intended by its designers. E.g., if the software for managing financial data is more reliable, its behavior is more consistent and its up time for end-users is *high*. If the software is less reliable, its behavior is less consistent and up time for end-users is *low*
Response time	Response time of software associated with an architecture measures how quickly the software responds to the end-user. E.g., if the software for managing financial data has a faster response time, then it will respond *faster* to the end-users. If the software has slower response time, then it will respond *slower* to end-users

Next, we describe how we refined external SQ into a list of factors, and the research methodology used to study their relative weights.

3. Identification of factors in the study

Any or all of the factors in Table 1 can conceivably affect the perception of SQ in the minds of senior IS managers. To identify the list of factors that are important, we adopted the following approach. First, we conducted an extensive literature review to identify the factors. Second, we conducted semi-structured interviews with randomly selected senior IS managers to ensure: (a) that we had not omitted any factor that was important in their decision models, and (b) that we operationalized the factors in terms that were understandable and familiar to the IS managers. We now describe both of these steps in detail.

After careful discussions between all three researchers involved in the project, we identified four factors that mapped reasonably well to those listed in previous literature. These are the features, [3] learnability, reliability and the response time of the software. These four factors are shown as columns in Table 3. The table provides a summary of how factors identified in previous literature map to the four factors we have identified in this work.

We now describe why we have excluded certain factors that are considered important in previous literature. *Maintainability* is frequently referenced as an element of software quality. Definitions range from error diagnosis, vendor service, repairs, enhancements, modification, and updating to satisfy new requirements (Barbacci et al., 1995; Boehm et al., 1978; Fenton, 1991; IEEE, 1992; Keller et al., 1990, 1995; QAI, 1989). *Expandability* and *flexibility* are included as elements of maintainability. We have chosen not to include maintainability in this study because this factor is inseparably tied with explicit cost (see Table 2). In (Bajaj, 2000), it was determined that explicit cost has much less weight than the other elements of software quality listed above. Furthermore, in (Kekre et al., 1995), which is the only study apart from the current one that looks at the relative weights of different factors on software quality, maintainability was shown to be less significant than other factors in driving software quality.

Installability is not included because Kekre et al. (1995) included it as an explicit factor and found its weights to be unimportant in their study. *Portability* is another frequently referenced factor. This factor refers to the compatibility of the system to different platforms, environments, or configurations (Boehm et al., 1978; Cardenas-Garcia, 1991; Christie, 1994; IEEE, 1992; Keller et al., 1990). *Reusability* has some similar characteristics, but also includes the ability to be easily converted for use in another application (IEEE, 1992; Keller et al., 1990). *Interoperability* is another related factor which refers to the degree to which the software can easily interface with another system (Keller et al., 1990). Finally, *structuredness* is an element of software quality referring to the organization of interdependent parts (Boehm et al., 1978). Bajaj (2000) studied *compatibility*, which was composed of the above factors (portability, reusability, interoperability, and structuredness) and determined it to be less important than the software quality when looking at the evaluation of computing architectures by senior IS managers.

Once we had identified a list of factors, we arrived at the definitions in Table 4 as follows. We randomly selected 12 organizations from a database of 232 large corporations in Pittsburgh, PA. The Chief Information Officers (CIOs) of seven of the 12 organizations agreed to be interviewed. In each semi-structured interview, which

[3] While features are not traditionally used when studying internal SQ, our literature review in Table 3 shows that they have been considered important in past work, when studying SQ as perceived by the users (external SQ). Our interviews with the CIOs confirmed that features should be included when studying SQ from their perspective.

lasted approximately 15 min, the CIO was asked to list the factors that he/she considered important from the point of view of his/her organization. The interviews were conducted for two reasons. First, we did not want to omit any factors considered important by senior IS managers, nor to include any factors that they considered irrelevant. Second, we wanted to describe the factors using terminology that was familiar to the CIOs, since the subjects in the next phase of the study would also be senior IS managers. The descriptions of the four factors in our study were determined by studying the field-notes of the interviews, previous literature and careful discussions amongst the three researchers involved in the study. Table 4 lists the factors and their definitions.

It is important to note that we were not seeking statistical validity when conducting the interviews. Instead, we were seeking *theoretical saturation* (Glaser and Strauss, 1967), a term understood in sociological theory to mean the gathering of data from subjects until, in the researchers' judgment, nothing new will be learned by gathering more data. None of the (randomly selected) senior IS managers we interviewed came up with new factors and they all used reasonably similar terminology when discussing the factors. In our judgment, interviewing more senior IS managers would have contributed nothing more towards arriving at the descriptions of the factors.

Next, we describe the second phase of the study: the estimation of the relative weights of these factors in the minds of senior IS managers.

4. The estimation of the relative importance of the factors constituting external SQ

In this study, we use conjoint analysis (CA), an experimental design and model building approach widely applied in marketing to evaluate new products (Green and Srinivasan, 1978, 1990) but relatively new to IS research. CA is derived from conjoint measurement theory (Luce and Tukey, 1964) – the study of functional relationships between multi-attribute stimuli and their subjective valuation. In CA, subjects directly rank and evaluate a set of products described by their factors, where the set of products and their factor levels are constructed as an orthogonal experimental design. This evaluation process is similar to real-world decision making. Then multivariate model estimation; for example, regressing product scores on the factor levels, yields weights for individual factors. In contrast, the alternative approach of multiattribute utility assessment requires that subjects directly assess tradeoffs between pairs of factors in terms of overall outcome or product utility (Keeney and Raiffa, 1976). Such assessments are cumbersome and far from the actual processes of decision makers.

In a typical CA study, the researcher first constructs a set of hypothetical products (in our case, softwares) by combining the possible attributes (or factors) at various levels for each attribute. The hypothetical products are presented to subjects, who provide an overall evaluation of each product, relative to the others (usually by giving each one a score). This corresponds to selection in the real world, where products are evaluated as a whole. All the overall scores provided by a subject are then decomposed, through multivariate estimation, to yield the relative importance of each of the factors in the decision model of that subject. Thus, CA yields a decision model at the individual level. The individual decision models can be checked for validity, by using a set of holdout products (products that are evaluated by the subject, but whose scores are not used to construct the decision model). The actual scores given by the subject for the holdout products can be compared with the predicted scores, to get a measure of the validity of the decision model.

While forming decision models at the individual level is powerful, even more powerful is the ability to aggregate these models to form an overall, statistically significant decision model for the population being studied, which we do in this study.

CA is advantageous in that first, subjects have to consider all attributes jointly (versus considering them in isolation for most other techniques) which necessitates a tradeoff between attributes (or factors), which is similar to real world decision making. Second, the relationship between attribute levels and the evaluation scores given by the subject can be non-linear (versus a linear assumption in most other techniques like linear regression or the analysis of variance). In fact, we can use CA to test whether the weight of a factor on the dependent variable (the score) is linear or not. Third, an individual decision model is created for *each subject* (versus merely collecting one data point for each subject) allowing the detection of inconsistent decision making in a subject. Fourth, all previous studies in the area, to the best of our knowledge, are post-hoc, which means that the users have to actually adopt and use a particular software and then evaluate it. Using post-hoc methods, it becomes harder to find users who have used the same software, and to also ensure random selection and other statistical controls. Furthermore, users in post-hoc studies may be biased for example, in supporting their decisions, rather than candidly evaluating products. In CA, since hypothetical products are used, subjects can be randomly selected and administered the same study, ensuring they evaluate the same products. CA thus allows for the generation of more valid decision models. In CA, it is important that the hypothetical products be *believable*, that the attributes be reasonably non-overlapping, and that the attributes each have approximately the same number of levels (Wittink et al., 1990). Since CA is a methodology

that is fairly novel to IS research, a more detailed description of CA, and issues such as hypothesis formulation and sample size are discussed in Appendix D.

The steps we followed in the CA study are outlined in Fig. 1. Next, we describe each of these steps in detail.

4.1. Identification of factors and creation of the study packet

After selecting the factors to describe software, which are shown in Table 4, the next step was to specify levels for each factor. In all cases, the levels chosen were *high*, *medium* and *low*, except for the *features* factor, which was either *sufficient* or *insufficient*. If we had used *high*, *medium* and *low* for *features* then both *high* and *low* features would have had negative connotations, since low would imply too few and high would imply too many (a features explosion). We constructed the following additive decision model for each subject:

$$\text{Software evaluation score} = \text{Features weight} + \text{Learnability weight} + \text{Reliability weight} + \text{Response time weight}. \quad (1)$$

The next step was to generate the orthogonal set of hypothetical softwares that would be evaluated by each subject to allow us to get the relative importance of each factor. The well known SPSS statistical package was used to generate the hypothetical softwares. Nine hypothetical softwares, each characterized by one level for each of the four factors were generated. In addition, we also generated four holdout softwares, to test the internal validity of the responses of each subject. Thus, each subject would be given the same 13 softwares. The 13 softwares are shown in Appendix A. An examination of the softwares indicates that none of the hypothetical softwares that are used are unrealistic in the real world, i.e., it is possible to imagine these softwares existing in the real world.

The third step was to operationalize the factors. Each subject was administered the study by the same researcher in person, and reliability and validity controls were implemented on site. Because of this, a richer operationalization of factors is possible here, than with a mail-out survey, where all the controls are part of the survey, since no verbal interchange can result between the researcher and the subject. For each factor we gave the definition and a reason why the factor was important. The reasons were carefully kept moderate, so as not to bias the subjects in favor of any factor. The idea behind the reasons was to simply highlight to the subject the effects of the extreme levels of each factor, and to achieve a *relatively uniform semantic range* amongst the subjects about what each factor meant. Table 4 lists the definitions and the reasons.

The fourth step was the construction and pilot testing of the study packet that would be used in the actual study. The 13 softwares were printed on separate cards, of identical length, breadth and thickness. We pilot tested the study with three doctoral students with high, moderate and low IS experiences, respectively. Based on the feedback, we made the following changes in the packet. Since the order of appearance of a factor on a card was important, we created four different study packets. Across the study packets, each factor showed up first in all the cards of one packet, second in all the cards of another packet, etc. Of course, the *same* 13 softwares were presented in each packet; only the order of factors describing each software on a card was changed across the four packets. The cards would be shuffled before being handed out to each subject, and the cards were titled from A–M, with the explicit mention to the subjects that the alphabets were chosen at random. Finally, the presentation (font size, etc.) on each card was identical. We also ensured that the operationalization of each factor was easily understood by all three pilot study subjects. Minor modifications were made based on insights gained from the pilot test. One final study packet (out of four) is shown in Appendix B.

We now describe how we ensured reliability and construct validity with each subject in the actual CA study. Each study was conducted with one subject, in the presence of the same researcher. The instructions in the packet asked the subject to read the descriptions of the factors. The next step in the study was for the researcher to answer any questions the subject may have regarding the descriptions of the factors, and to ensure that the subject had an understanding of how each factor was different from the other. This dialogue with the subjects was necessary to ensure that all subjects had a similar understanding of the four factors. At this stage, they were also asked if, in their opinions, any important factors had been omitted. This was an added, informal check on whether our factors were complete. [4] Once the researcher was satisfied that the subject had a good

1. Identify factors important in the decision space of IS managers when evaluating software used in their organization.

2. Select appropriate levels for each factor (attribute).

3. Operationalize each factor in a manner suitable for a face-to-face study.

4. Create study packet and pilot test for clarity of measures, time taken for one study, any other implementation problems or possible biases.

5. Select subjects.

6. Administer the study to each subject individually, in the presence of the researcher.

7. Analyze data, come up with individual decision models for each subject, as well as an aggregate decision model across the sample, and present results.

Fig. 1. List of steps that would constitute a CA study in IS.

[4] All of the subjects in the study, described next, indicated that the four factors covered their decision space.

Table 5
Response details of firms contacted

Agreed to study	Contacted but did not return calls	Declined to participate	Total number of organizations contacted
24	9	11	44

understanding of the different factors, the subject was asked to rank order the cards in descending order of preference. No time limit was to be set for the ranking, and it typically was expected to take between 20 and 30 min to perform the ranking. Once the cards were rank ordered, the subject was to give a score of 100 to the highest card, and one to the lowest card. The remaining cards would each be given any score, as long as a strict order was maintained. These scores would be the (metric) dependent variable in the study, and would represent the evaluation score of the software shown on that particular card, by the subject, for their organization.

4.2. Subjects for the study

The population for our study consisted of a database of 232 firms [5] located in Pittsburgh, PA. We selected a random sample of 44 corporations from this population, and identified the CIO or senior IS manager in each corporation. We made sure that these managers were decision-makers in terms of making significant new investments in IS within the organization. The CIOs were contacted, and 24 agreed to participate in the study. The details of the response rate (55%) are shown in Table 5.

The demographics of the 24 senior IS managers who participated in the final study are shown in Table 6. The table indicates the wide variety of organizations represented in the random sample.

4.3. Data analysis

In our case the dependent and independent constructs were metric. Hence, we used dummy variable (categorical variable) regression analysis (using the well known Excel package) to estimate a part-worth model for each subject (each IS manager). Internal validity in a CA study translates to whether each subject's decision model represents a consistent logic or not. Internal validity of each individual subject's model was tested based on the holdout sample of four observations for each subject. The Wilcoxon rank test [6] (Wonnacott and Wonnacott, 1984, pp. 472) was used for this. The test ranks predicted values and actual values and then answers the question: are the two populations significantly different from each other? In all 24 cases, the IS managers had valid internal decision models.

Based on the dummy variable coding scheme for the nine softwares (as represented by the factors) we used, the part-worth estimates are on a common scale. Hence, the overall relative importance of each independent factor for a subject can be easily computed by looking at the range of part-worths across the levels of that factor.

4.4. Results

The expected relative part-worth of each factor is 25% (since there are four factors, occupying 100% of the subjects' decision space). We use two metrics to present the results. The first metric is *the mean relative part-worths* of each of the four factors, and the *confidence intervals* of these means. This metric is equivalent to testing a null hypothesis that all four factors have an equal weight in the minds of the senior IS managers. Since the mean part-worth can be biased by extreme values in the sample, we use a second metric, which gives *the percentage of subjects in the sample* that indicated a higher than the expected 25% relative part-worth for each of the four factors. Table 7 shows the mean relative part-worths and confidence intervals for each factor (the first metric). It also depicts the percentage of subjects in whose decision model the factor had a part-worth over the expected 25% (the second metric). In addition, Table 7 lists the direction of influence in the cases that were significant and the linearity of effect of the factor on the dependent variable (the scores). Note that to estimate linearity, at least three levels are needed, so the *features* factor's linearity of effect cannot be estimated. The data and figures used for the results are in Appendix C.

From Table 7, it is clear that *reliability* has the highest mean relative worth of any factor. Its confidence interval does not overlap any of the other factors, thus disproving the null hypothesis. Using the second metric also, it is clear that *reliability* is the most preferred. Nineteen of the 24 participants (79%) gave this factor greater than expected weight (25%).

Response time appears to be the next most important factor, though its confidence intervals overlap with *features* and *learnability*. On the second metric as well, *response time* is second to reliability. *Learnability* and *features* are the least important, being approximately equal on the first metric, and with *features* being more important than *learnability* on the second metric.

All four factors have an (expected) positive slope (implying that more is preferable to less), and a linear slope where the number of levels is greater than two. The linearity finding in our study implies that future research using these factors can assume a linear effect of these factors on software evaluation.

[5] Each corporation in this database has over 250 employees.

[6] An analysis of variance could not be used, since the number of observations in the holdout sample is small (four observations each). A larger holdout sample, which could have cognitively overloaded the subjects, thus leading to serious biases in their responses.

Table 6
Demographics of IS managers who participated

Subject no.	Number of machines	Years of experience	Education	Gender	Environment most comfortable managing	SIC (Standard Industrial Classification)[a]
1	400	10	HS	M	Client/server	3312; 3356
2	1200	6	BS	F	Fully distributed, client/server	13; 49
3	600	15	BS	M	Client/server	5141; 5411
4	40	25	BS	F	Client/server, mainframe	3312; 3462; 3643
5	1200	10	BS	M	Fully distributed, client/server	99
6	2	23	AA	F	Mainframe, client/server	2751; 2262; 2641
7	75	10	AA	M	Mainframe	2821
8	500	6	MS	M	Client/server	3845
9	300	35	AA	M	Mainframe	3312; 5812
10	2000	24	BA	M	Mainframe	4923
11	100	15	BS	M	Mainframe	3621
12	220	15	AA	M	Client/server	3444; 8711; 8748
13	200	11	BS	M	Client/server	2821
14	735	28	BS	M	Mainframe, client/server	99
15	400	1	BS	M	Client/server	2711; 2752
16	350	10	BS	M	Intranet	3679
17	23,000	20	BA	M	Mainframe, client/server, fully distributed, intranet	6025
18	650	30	MS	M	Mainframe, client/server	3255
19	125	24	AA	M	Client/server	3317; 3499
20	110	3	AA	M	Client/server	3325
21	200	30	BS	M	Fully distributed	3317
22	100	11	BS	M	Mainframe, client/server	5051
23	250	30	BS	M	Client/server	99
24	35	8	BA	M	Client/server	89

[a] This information is shown to demonstrate that our sample set was indeed varied. The 20 architectures in the study were all hypothetical, and this was explained to the IS managers.

Table 7
Summary of results for each factor across the sample

	Features	Learnability	Reliability	Response time
Mean	15.10	14.60	46.82	23.47
Confidence interval (95.0%)[a]	9.57–20.63	9.24–19.96	39.12–54.52	15.96–30.98
Percentage of respondents who considered significant[b]	25%	8%	79%	38%
Direction of slope[c]	Positive	Positive	Positive	Positive
Linearity of slope[d]	NA	Linear	Linear	Linear

[a] Degrees of freedom = 20.
[b] This is the percentage of subjects for whom the relative part-worth was >25% for this factor.
[c] This is the direction of the slope of the line only for those subjects for whom the factor had a relative part-worth >25%.
[d] Only for those subjects for whom the factor's relative part-worth >25%.

4.5. Discussion

The surprising finding in our study is that *features* and *learnability*, which have been found to be most important in earlier studies (Kekre et al., 1995), are considered much less important for the software used in large organizations today, at least by senior IS managers who have several years experience with implementing systems and dealing with end-user issues. Our study shows that the factor that is dominant in their decision models today is reliability, followed by response time. These findings have implications for IS theory, for IS curricula and for IS practice.

In IS theory, this is the second work, to the best of our knowledge, that examines the relative weights of the factors that make up software quality of off-the-shelf software. The earlier study (Kekre et al., 1995) found that, once an acceptable level of reliability was achieved, the *feature set* (capability) was a strong factor, along with the *learnability* and *memorability* (both called usability) of the software. Another factor that was found to have reasonably strong influence was the response time of the software (performance). Our results indicate that the nature of software has changed since the time of the study by (Kekre et al., 1995). *Learnability* and *feature set*, which were found to be

the primary drivers there are no longer important. The reliability [7] of the software is of primary concern to end-users. This result is intuitively supported by the changing nature of software. Thus, today, almost all software is increasingly graphical user interface (GUI) based, with extensive on-line help availability, and with a multitude of features, often many more than are required. With software of this type being used increasingly, it appears that learnability and features are not important in the minds of the consumers of this software. Our study indicates that software used in large organizations has evolved to the point where it is, in general, rich enough in *features* and easy enough to *learn* and *remember* to use, but still lacking comparatively in *reliability*. Essentially, combining our findings with (Kekre et al., 1995) indicates that user's perceptions about software change over time. A second contribution to IS theory is that, to the best of our knowledge, this is the first work that uses conjoint scaling and analysis for analyzing the trade-offs regarding software quality. The methodology we use is replicable and can potentially allow for findings across multiple studies. A third contribution is the finding that the effects of all factors in the study (with more than two levels) is *linear*, which gives future research studies justification for making a linearity assumption when modeling the effects of these factors, and allowing the use of statistical techniques like linear regression.

The findings of this study provide directions for the design of future IS curricula. The high importance of reliability indicates a clear need in the IS and computer science curricula for more courses that teach both the theoretical aspects of software reliability e.g., (Best et al., 2000; Perry et al., 2000), as well as practical methods to create reliable software e.g., (Bachmann et al., 2000). These courses should be in addition to the typical software engineering courses already taught in most curricula.

Our findings are also important for IS practice. First, our findings indicate that developers of off-the-shelf software used in corporations will gain more competitive advantage by focusing on improving the reliability of their software, than by improving the learnability of or the number of features in their software, given currently used software as a baseline. This also holds true for IS consulting firms, who have a software or a turnkey solution as a deliverable to their clients. Second, building on the results of an earlier study by (Bajaj, 2000), the findings of this study clearly point out potential avenues of competitive advantage for proponents of new computing architectures, such as the network computer architecture. *External software quality* is the most important factor in the decision models of senior IS managers when evaluating computing architectures, and the reliability of software is the area to focus on if a higher evaluation of an architecture is desired by its proponents. For example, proponents of the network computer architecture would be best served if they focused on improving the comparative reliability of the software that is developed on the architecture versus software available on existing architectures like the client/server architecture (assuming that the learnability and feature set are comparable). The secondary importance of response time in our study implies that network latencies and other causes of poor response time are also areas to concentrate on, in order to improve evaluations. Third, the findings here have implications for researchers in the area of software reliability as well. While software reliability has been extensively studied in literature e.g., (Goel, 1985), there is a clear need for researchers to communicate this research to industry, where off-the-shelf software is built and used by the subjects of our study. Thus, methodologies like the *architecture based design method* (Bachmann et al., 2000), that lay down practical rules that project managers can follow to engineer reliability at the design phase itself, are a step in the right direction.

4.6. Limitations

The methodology used in this study has limitations. The face-to-face method of data collection used in our study allows for a richer operationalization of each factor (see Appendix B) than is allowed in a mail-out survey, since it is possible to clarify issues related to the study to subjects at the time of administration. However, by the same token, the method is highly researcher-dependent, and the potential to bias subjects one way or another is certainly higher than with a mail-out survey, where subjects do not see the researcher. Second, a richer operationalization allows us to consider more realistic factors, but the construct validity is harder to quantify, as opposed to standard techniques like the Cronbach alpha which are available for Likert scale type questions used in mail-out surveys, that are more traditional in IS research.

5. Conclusion

In this work, we used previous literature to refine the external SQ (i.e., the quality from the point of view of the users of the software) of software used in large organizations, into a set of four factors. We interviewed senior IS managers of seven randomly selected large corporations to operationalize the factors in terms familiar to the managers. Next, we conducted a face to face conjoint analytic study, using an additive model, to evaluate the individual decision models of senior IS

[7] Recall that reliability is the extent to which the software behaves as intended by its designers.

managers of 24 randomly selected corporations. After testing each of these models for internal validity, using a holdout sample, we aggregated these models to test the null hypothesis that the relative weight (part-worth) of each of the four factors in a population level decision model is equal. The null hypothesis was disproved for the *reliability* factor, clearly indicating that it is dominant in the decision model of the population, with the *response time* factor being of secondary importance. These findings contradict those of studies conducted earlier, where the *learnability* factor and the *feature* set factor was most important. The findings in our study indicate that the nature of software used in corporations has changed, and that software developers and consulting firms should focus on methods of developing reliable solutions. Designers of IS curricula should allocate resources towards developing courses on reliability evaluation, and researchers in the area of software reliability need to communicate their knowledge to industry.

In terms of future research, the results of this study clearly point to the construction of reliability benchmarks for end-user software as well as all the software it rests on (such as operating systems and database management systems). Thus, benchmarks may be made for end-user software such as word-processors, spreadsheets and e-mail packages, as well as for operating systems and database management systems. This will give IS managers more information that is important to their evaluation of both software as well as computing architectures for their organization.

Acknowledgements

The authors thank Dr. Steve Cross, Director of the Software Engineering Institute, Carnegie Mellon University, the editor-in-chief as well as anonymous reviewers, all of whose comments greatly improved the quality of this paper.

Appendix A

See Table 8.

Appendix B. Copy of study packet

Directions

Thank you for participating in our study. Your cooperation is greatly appreciated and crucial to the success of this study.

Please be sure to answer all of the questions as your responses will only be useful if they are complete.

The results of this study will indicate which elements of software quality are usually considered by IS managers, like yourself, when making decisions regarding significant new computer purchases. The results will likely be interesting to IS managers like yourself. A complimentary copy will be mailed to you, once the study is completed.

Your responses will be kept confidential, and available only to the researchers actually conducting the study. Please feel free to call me at any time if you have any questions.

Bonnie Brinton Anderson,
Heinz School of Public Policy and Management,
Carnegie Mellon University,
4800 Forbes Avenue,
Pittsburgh, PA 15213.
bbrinton@andrew.cmu.edu
(412) 268-1415

Table 8
List of hypothetical architectures (information from each of the cards)

Architecture	Features	Learnability	Reliability	Response time
A	Less	Medium	Medium	Average
B	Sufficient	Medium	High	Slow
C	Sufficient	Low	Medium	Fast
D	Sufficient	High	Low	Average
E	Less	Medium	Low	Fast
F	Less	High	Medium	Slow
G	Less	Low	Low	Slow
H	Less	High	High	Fast
I	Less	Low	High	Average
J	Less	Low	High	Slow
K	Sufficient	Low	High	Average
L	Sufficient	Low	High	Slow
M	Less	High	Low	Slow

The 13 hypothetical computing architectures (9 for the orthogonal set and 4 holdout) generated by SPSS.

Demographic information

Name:

Organizational Address:

Organizational Position and Duties:

Numbers of Machines Managed:

Years of Experience in the IS Area:

Highest Educational Degree:

Which best describes the computing environment you feel most comfortable managing (circle one, please):
Mainframe-based systems
Client server systems
Intranet-based systems
Fully distributed systems

Please read the following information carefully in order to understand the study.

This study looks at what software quality factors IS managers, like yourself, consider when selecting computing architectures for your organization. There are several computing architectures that are available. Examples of computing architectures include the following:

- mainframe systems with terminals,
- client server systems (client and server machines dividing up the processing),
- the proposed architecture of diskless network computers running off an intranet server, and
- a fully networked architecture where each machine is a server by itself, and communicates with every other machine.

A computing architecture consists of both hardware and software. A shift to another architecture can have a profound effect on how organizations perform their business.

In this study, we will assume that the quality of the general software associated with a computing architecture is *completely described* by the following factors:

1. *Features* of a software support what end-users do with the software. E.g., there is a certain set of core tasks that end-users do with financial data, and software for managing this would have features that support these core tasks. If a software has less features then some core tasks will not be supported. If a software has sufficient features, then all core tasks will be supported. In this study, the software associated with a computing architecture can have less features or sufficient features.
2. *Learnability* of a software measures how easy it is for end-users to learn to use the software and to remember how to use the software. E.g., if a software for managing financial data has higher learnability, then it will be easier for end-users to learn to use it and to remember how to use it. If the software has lower learnability, then it will be less easy for end-users to learn to use it and to remember how to use it. In this study, the learnability of the software associated with a computing architecture can be high, medium, and low.
3. *Reliability* of a software is the extent to which it functions as intended by its designers. E.g., if a software for managing financial data is more reliable, its behavior is more consistent and its up-time for end-users is higher. If the software is less reliable, its behavior is less consistent and up-time for end-users is lower. In this study, the reliability of the software associated with a computing architecture can be high, medium or low.
4. *Response time* of a software measures how quickly the software responds to the end-user. E.g., if a software for managing financial data has a faster response time, then it will respond faster to the end-users. If the software has slower response time, then it will respond slower to end-users. In this study, the response time of the software associated with a computing architecture can be fast, average, or slow.

You will now be presented with descriptions of software associated with 13 different computing architectures. These architectures do not have names, but are arbitrarily labeled from A to M. The software quality of each computing architecture will be completely described by the four factors we have discussed.We would like you, as a senior IS manager, to do the following:

- Please sort these 13 architectures (on the 13 different cards) in descending order of preference (from most preferred on the top of the pile to least preferred at the bottom).
- After you have sorted the cards, please write a number on each card that gives a numerical value to your preference, from 1–100. The least preferred architecture (at the bottom of the pile) will be given a score of 1, while the most preferred architecture will be given a score of 100. The cards in between should be given a preference score between 1 and 100. Each card should have a preference score lower than the card below it. *However*, *the scores need not be spaced equally*. It is entirely up to you to choose the score you wish to give each architecture. Note that the entire architecture should be given one preference score, based on how appealing it is to you.

Also, in case you change your preferences, you may reorder the cards in the heap at any time during the study. If you do alter the order, please make sure you alter the preference scores as well, i.e. the preference

score of every card is still between the scores of the cards above and below it.

Since we shall be re-using the cards, please use the pencil provided to write on the cards. All the factors discussed earlier have been summarized on a single sheet for your convenience. Please feel free to refer to this. Below is an example of one CA on a card. In all, the packet had 13 cards, one for each CA. This packet is one of four packets. The other packets list the factors in a different order.

Architecture A

Features: *Less*

Learnability: *Medium*

Reliability: *Medium*

Response time: *Average*

Appendix C

See Tables 9–11, and also see Figs. 2–5.

Appendix D

D.1. Developing a CA based methodology for IS studies

CA is related to traditional experimentation, in which the effects of levels of independent variables are determined on a dependent variable. E.g., the effects of temperature and pressure on the density of soap in a soap manufacturing process. In situations involving human behavior, such as in IS, we want to also determine the effects of levels of certain variables (equivalent to independent variables) on the dependent variable, which is often an overall rating or a purchase decision. However, the "independent variables" in human behavior studies are often weakly measured or qualitatively specified (Green and Srinivasan, 1978). An example in IS would be whether a system is decentralized or centralized, and the effect of this variable on an overall evaluation (the dependent variable).

The basic model in a CA study is:

$$Y_1\ (\textit{metric or non-metric}) = X_1 + X_2 + X_3 + \cdots + X_n\ (\textit{non-metric})$$

(Metric refers to an interval or ratio scale, while non-metric refers to a nominal or ordinal scale.)

The main advantages of CA from a statistical perspective, are its ability to accommodate metric or non-metric dependent variables, its ability to use non-metric variables as predictors and the quite general assumptions about the relationships of the independent variables with the dependent variable (e.g., no linearity assumptions are made) (Hair, 1992). A CA study has two main objectives. First, to determine the contributions of various predictor variables (also called attributes) and their respective values (or levels) to the dependent variable (usually an overall evaluation of a product or concept), and second, to establish a predictive model for new combinations of values taken from the predictor variables.

CA is based on the premise that subjects evaluate the value or utility of a product/service/idea (real or hypothetical) by combining the separate amounts of utility provided by each attribute. CA is a decompositional technique, because a subject's overall evaluation is decomposed to give utilities for each predictor variable, and indeed for each level of a predictor variable. The overall relative utility for each predictor variable or attribute is called the part-worth of that attribute. CA is common in behavioral studies (Luce and Tukey, 1964) and in marketing studies (Green and Rao, 1971), where the predictor variables are often called attributes, and the dependent variable is often an overall evaluation of a product.

Table 9
Dummy variable coding of architectures

Architecture	Features sufficient	Learnability medium	Learnability high	Reliability medium	Reliability high	Response time average	Response time fast
A	0	1	0	1	0	1	0
B	1	1	0	0	1	0	0
C	1	0	0	1	0	0	1
D	1	0	1	0	0	1	0
E	0	1	0	0	0	0	1
F	0	0	1	1	0	0	0
G	0	0	0	0	0	0	0
H	0	0	1	0	1	0	1
I	0	0	0	0	1	1	0
J	0	0	0	0	1	0	0
K	1	0	0	0	1	1	0
L	1	0	0	0	1	0	0
M	0	0	1	0	0	0	0

Table 10
Dummy variable coefficients for each participant

Participant	FL	FS	LL	LM	LH	REL	REM	REH	RTL	RTM	RTH
1	0	8.67	0	5.67	9.00	0	34.67	72.00	0	3.67	8.00
2	0	−0.17	0	6.33	8.00	0	28.00	71.33	0	9.67	19.67
3	0	10.67	0	6.33	6.33	0	34.67	78.00	0	8.00	14.67
4	0	4.17	0	14.67	14.00	0	35.33	70.33	0	9.00	14.67
5	0	11.50	0	8.00	18.00	0	29.67	71.33	0	1.33	9.67
6	0	16.17	0	8.00	−2.33	0	23.33	63.33	0	−12.33	−2.00
7	0	11.50	0	12.67	9.33	0	29.67	59.33	0	6.67	12.33
8	0	10.17	0	−7.67	1.33	0	27.00	68.67	0	9.67	29.00
9	0	20.17	0	4.67	14.00	0	30.67	63.00	0	3.67	22.00
10	0	22.00	0	8.33	11.67	0	36.00	48.00	0	9.67	9.33
11	0	12.00	0	15.67	18.33	0	35.00	55.00	0	14.33	25.67
12	0	15.50	0	−12.67	−9.33	0	23.67	56.33	0	18.00	32.00
13	0	26.50	0	0.33	−13.33	0	30.33	53.67	0	3.33	18.67
14	0	1.67	0	5.67	18.00	0	33.00	45.67	0	15.33	35.33
15	0	−1.00	0	34.67	46.33	0	49.67	36.33	0	4.67	16.33
16	0	35.00	0	7.67	−8.67	0	28.33	51.67	0	−12.00	6.00
17	0	8.67	0	−13.67	16.67	0	23.33	49.67	0	19.33	32.67
18	0	35.00	0	5.67	5.33	0	26.67	42.33	0	12.67	41.33
19	0	45.67	0	9.67	8.00	0	26.33	36.33	0	3.00	4.67
20	0	44.50	0	−10.67	−14.33	0	32.33	27.67	0	6.33	19.67
21	0	−0.33	0	2.67	2.00	0	12.33	23.33	0	39.00	73.67
22	0	2.33	0	4.67	−0.33	0	13.00	21.33	0	46.33	78.00
23	0	1.67	0	24.00	63.67	0	−2.33	17.00	0	2.33	18.33
24	0	39.17	0	−13.67	9.33	0	10.67	7.00	0	19.00	52.67

Several works highlight CA in detail (Hair, 1992; Luce and Tukey, 1964; Wittink et al., 1990). Without substituting for them in any way, we present a simple description here of the essential concepts in a CA study. For a CA study, a product class is considered, along with a set of subjects who would evaluate products in that class. A set of attributes (predictor variables) is selected to describe the product class. The possible levels of each attribute are selected. A product in the product class is then simply a combination of attribute levels (one level per attribute).

Table 11
Relative part-worths for each subject

Participant	Features	Learnability	Reliability	Response time
1	8.87	9.22	73.72	8.19
2	0.17	8.07	71.93	19.83
3	9.73	5.78	71.12	13.37
4	4.01	14.13	67.74	14.13
5	10.41	16.29	64.56	8.75
6	15.82	10.11	61.99	12.07
7	12.00	13.22	61.91	12.87
8	8.70	7.70	58.77	24.82
9	16.92	11.75	52.87	18.46
10	24.09	12.77	52.55	10.58
11	10.81	16.52	49.55	23.12
12	13.30	10.87	48.35	27.47
13	23.56	12.15	47.70	16.59
14	1.66	17.88	45.36	35.10
15	0.88	40.88	43.82	14.41
16	28.93	13.50	42.70	14.88
17	21.43	13.23	39.42	25.93
18	47.40	10.03	37.72	4.84
19	28.15	4.56	34.05	33.24
20	40.15	12.93	29.17	17.74
21	0.33	2.67	23.33	73.67
22	2.19	4.69	20.00	73.13
23	1.66	63.25	16.89	18.21
24	31.21	18.33	8.50	41.97

The method of data collection in the CA study can be face-to-face, which is more time consuming, but allows for a richer operationalization of each attribute, or by mail, which allows for greater reach of subjects but permits leaner operationalizations in the interests of validity. A face-to-face data collection method, such as used in the current study, represents potentially a happy medium between a case study (where the operationalization is very rich but validity is often criticized) and a simple Likert scale survey questionnaire, where the operationalization is very lean, though validity is quantifiable, using techniques like factor analysis and Cronbach's alpha (Nunnally, 1978). The method of data analysis depends on whether the dependent variable is metric (in which case categorical variable regression can be used) or non-metric (in which case logistic regression or discriminant analysis can be used). A further choice facing the researchers is the composition rule to be used: additive or with interactive effects. For most situations where a predictive model is desired, and where the attributes involve less emotional or aesthetic judgments and are tangible (as is reasonable to assume in IS) an additive model is usually sufficient (Hair, 1992).

From an application perspective, the CA methodology has several advantages. First, it permits the con-

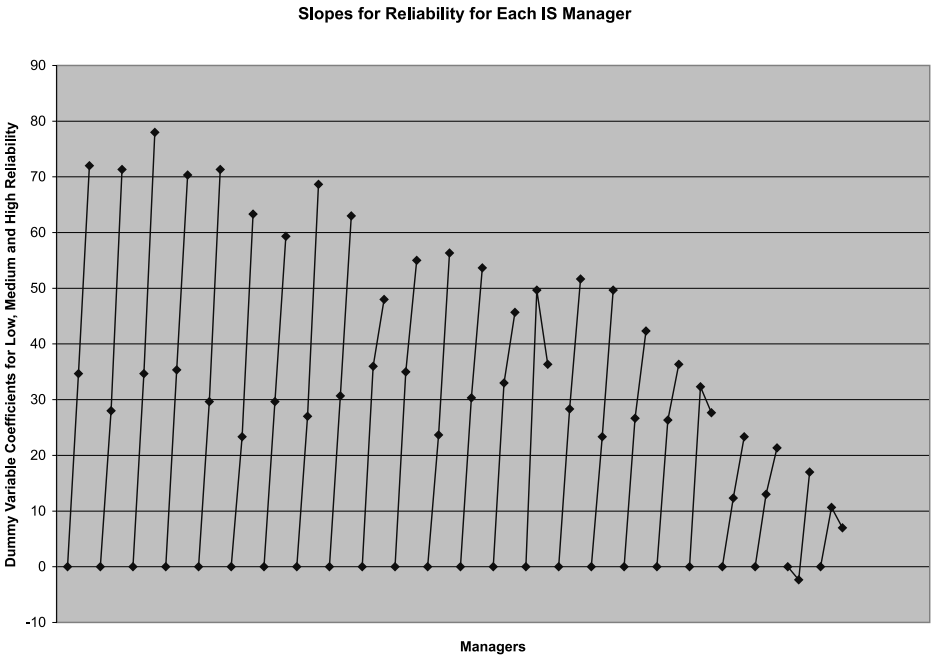

Fig. 2. Dummy variable coefficients for reliability.

struction of utility models in application areas where the predictor variables are often weakly quantifiable, as in the case of studies involving perceptions, which are commonplace in IS research.

Second, a CA study allows for a more realistic overall decision model for a population, because it forces subjects to evaluate the products as a whole (as in real life). It forms individual decision models for each subject, that can be tested for internal validity by using a hold out sample (a set of products in the product class whose predicted evaluations are compared with the subject's actual evaluations); and it allows the formation of an

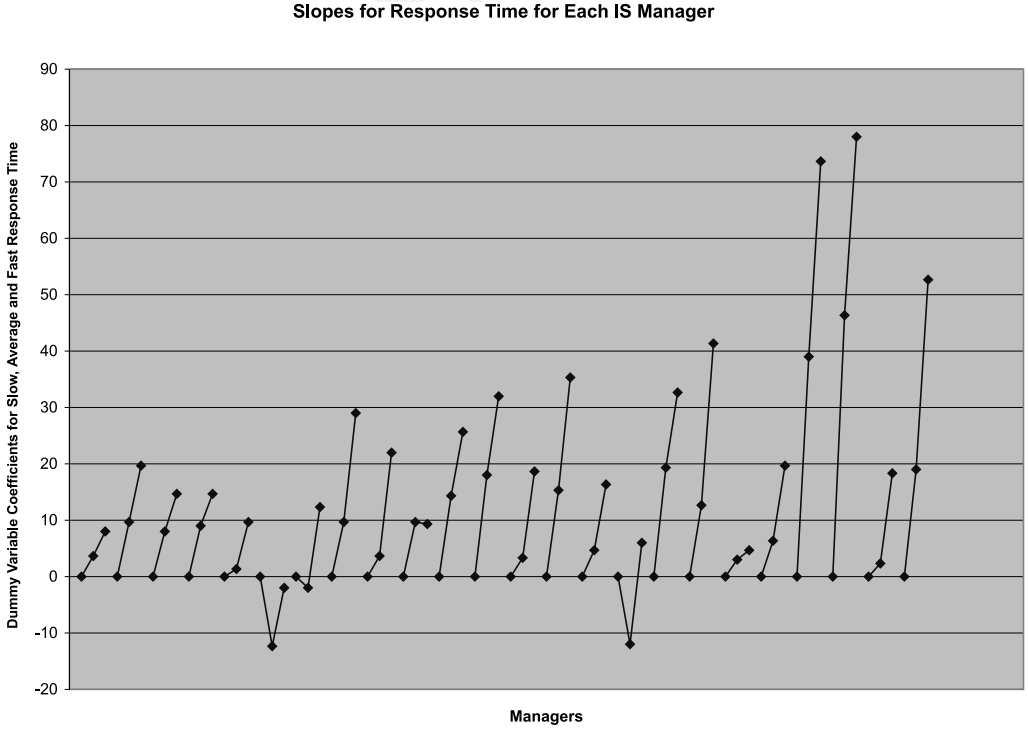

Fig. 3. Dummy variable coefficients for response time.

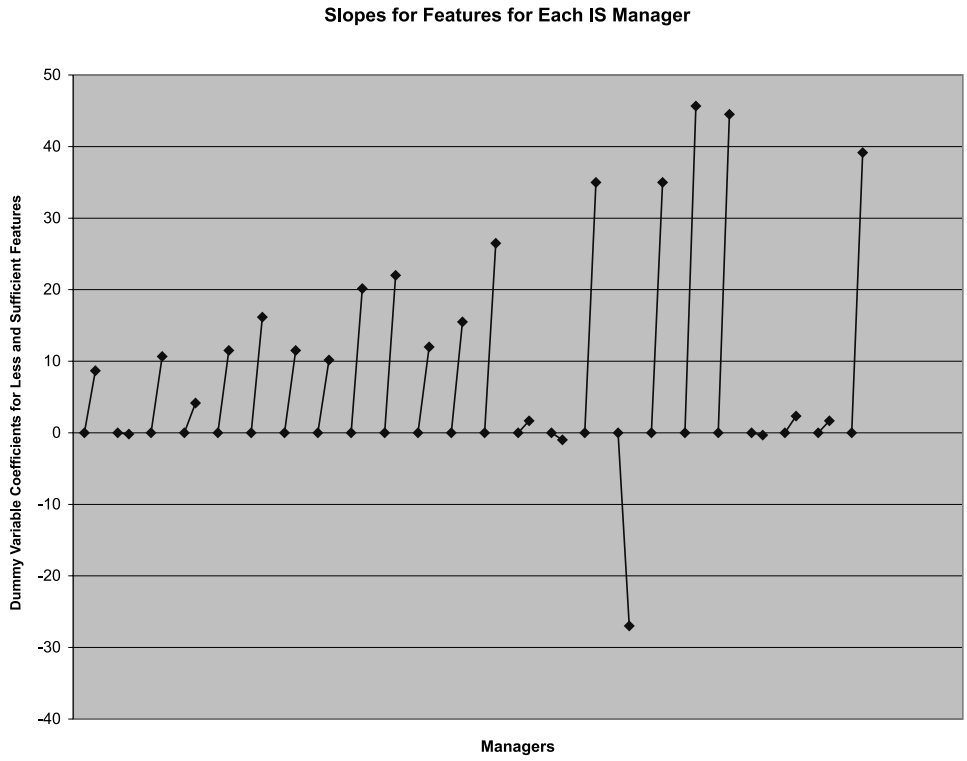

Fig. 4. Dummy variable coefficients for features.

aggregate decision model across all the subjects, and permits the statistical testing of the null hypothesis that all the attributes have an equal utility in the aggregate decision model.

Third, CA makes no assumptions about the nature of the relationships between the attributes and the dependent variable. This makes it very useful when exploring unknown variables as potential predictors.

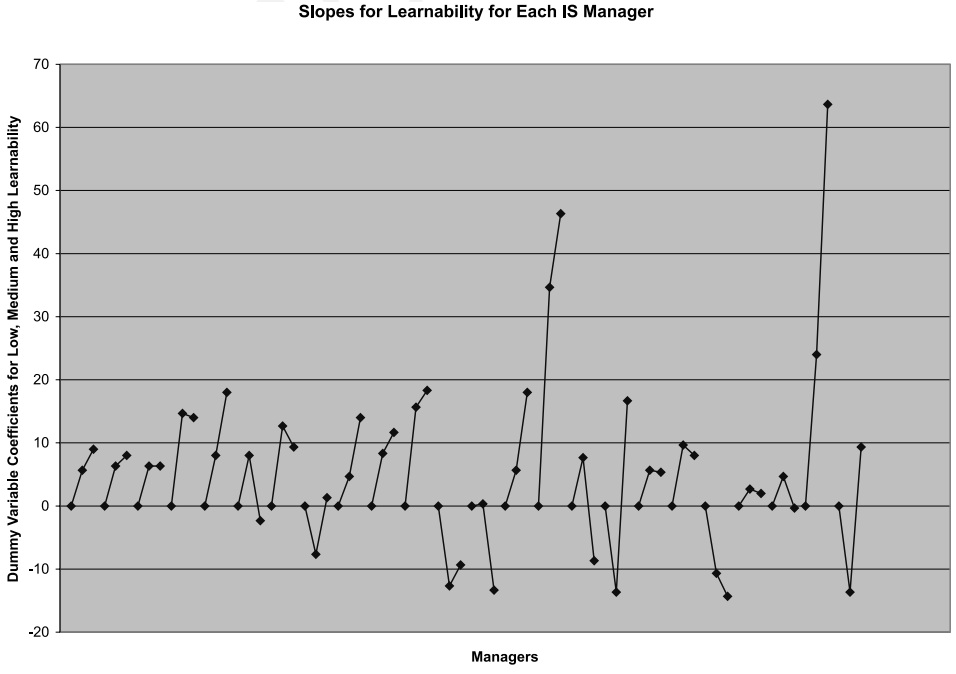

Fig. 5. Dummy variable coefficients for learnability.

D.2. Operationalizing and selecting levels and scales for the predictor variables (Atributes)

The responses in a CA study are very dependent on the way the attributes and the scales (the number of levels and the range of the levels for an attribute) are presented to the subjects. If attributes are chosen that are prima facie known to be of less importance than others, then that will certainly affect the outcome. So, if we know before hand that, let us say, *Reliability*, as defined and scaled for the subjects is not likely to be as important as, let us say, *Learnability*, as defined and scaled, then that is probably what the outcome will be. What is needed in a study that seeks to assess relative part-worths of each attribute, is to operationalize the attributes (which are qualitative concepts) in such a way that their importance for the subjects are prima facie the same, as they are presented and scaled in the study. This will allow the study to be conducted as a classical hypothesis test, with the null hypothesis being that the relative part-worths of all attributes (predictor variables) as they are scaled, are equal.

Another issue with operationalization deals with construct validity: i.e., first, do all the subjects have a reasonably consistent idea of each attribute and its scaling, and, second, is this idea the same as what the researchers think it is. So a faulty operationalization will leave different subjects interpreting the constructs (or attributes) differently, while a better operationalization will mean that different subjects view the attributes and their scales in the same way.

One way to ensure construct validity and allow realistic scaling, is to ask a sample in the subject population itself to define the predictor variables. This technique allows the researcher to define the predictor variables (attributes) in a manner uniformly understandable to the subjects, and to also identify realistic end-points of the scales used for the attribute levels. This has been done in this study.

D.3. Hypothesis testing and sample size issues in a CA study

As mentioned in Section D.2, the CA study can be constructed as a classical hypothesis test, with the null hypothesis being that the part-worths of all the attributes are equal. In order to test such a hypothesis, we proceed as follows. First, individual decision models for each subject in the sample are constructed. These individual decision models give the part-worths of each attribute, for each subject. In this study, Section 4 in Appendix C shows this information. Once the part-worths of each subject in the sample are obtained, they can be aggregated to get a mean part-worth for each factor, for the sample. The mean value and the variance are then sufficient to statistically test the null hypothesis. The regular caveats of using too large a sample size apply. Thus, several basic statistical text books on hypothesis testing e.g., (Wonnacott and Wonnacott, 1984) caution against using too large a sample size, because that would indicate statistical validity for even small differences in means; differences that may not be actually significant for the situation under study. The sample size [8] is closely related to the degrees of freedom in the test, and a small sample size indicates fewer degrees of freedom, leading to a wider confidence interval. Thus, statistical validity from a smaller sample size (as long as the sample is random) is a good indicator that some real differences in the means have been found. In this study, we use a sample size of 23, and obtain statistically valid differences between some of the means (thus disproving the null hypothesis of our study).

The steps to be used in a CA study for an IS are summarized in Fig. 1.

References

Bachmann, F., Bass, L., Chastek, G., Donohoe, P., Peruzzi, F., 2000. The Architecture Based Design Method, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, CMU/SEI-2000-TR-001.

Bajaj, A., 2000. A study of senior information systems managers decision models in adopting new computing architectures. Journal of the Association of Information Systems 1 (4).

Barbacci, M.R., Klein, M.H., Longstaff, T.H., Weinstock, C.B., 1995. Quality attributes. SEI, Carnegie Mellon University, Pittsburgh, CMU/SEI-95-TR-021.

Barbacci, M.R., Klein, M.H., Weinstock, C.B., 1997. Principles for evaluating the quality attributes of a software architecture. SEI, Carnegie Mellon University, Pittsburgh, CMU/SEI-96-TR-036.

Bays, M., 1995. Impact of IS alignment strategies on organizational perceptions of quality. In: Proceedings of the 28th Annual Hawaii International Conference on System Sciences, Hawaii, USA.

Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merritt, M.J., 1978. Characteristics of Software Quality. American Elsevier, New York.

Cardenas-Garcia, S.R., 1991. A formal framework for evaluating multiattribute software specifications. Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park.

Christie, A.M., 1994. A practical guide to the technology and adoption of software process automation. SEI, Carnegie Mellon University, Pittsburgh, CMU/SEI-94-TR-007.

Cusumano, M.A., Kemerer, C.F., 1990. A quantitative analysis of US and Japanese practice and performance in software development. Management Science 36 (11), 1384–1406.

Best, E., Devillers, R., Koutny, M., 2000. Petri Net Algebra Monographs in Theoretical Computer Science. Springer, Berlin.

Fenton, N., 1991. Software Metrics: A Rigorous Approach. Chapman & Hall, London.

Florac, W.A., 1992. Software quality measurement: a framework for counting problems and defects. SEI, Carnegie Mellon University, Pittsburgh, CMU/SEI-TR-22.

Glaser, B.G., Strauss, A.L., 1967. The Discovery of Grounded Theory. Aldine, Chicago, IL, USA.

[8] We are assuming a random sample here.

Goel, A., 1985. Software reliability models: assumptions, limitations and applicability. IEEE Transactions on Software Engineering SE-11, 1141–1423.

Green, P.E., Rao, V.R., 1971. Conjoint measurement for quantifying judgmental data. Journal of Marketing Research 8 (August), 355–363.

Green, P.E., Srinivasan, V., 1978. Conjoint analysis in consumer research: issues and outlook. Journal of Consumer Research 5 (September), 103–123.

Green, P.E., Srinivasan, V., 1990. Conjoint analysis in marketing: new issues with implications for research and practice. Journal of Marketing 54 (4), 3–19.

Hair, J.F., 1992. Multivariate Data Analysis with Readings, third ed. Macmillan, New York, NY, USA.

Halstead, M., 1977. Elements of Software Science. Elsevier, North-Holland, New York.

IEEE, 1992. Standard for a software quality metrics methodology. IEEE, Standard 1061–1992.

Keeney, R.L., Raiffa, H., 1976. Decisions With Multiple Objectives: Preferences and Value Tradeoffs. Wiley, New York.

Keller, S.E., Kahn, L.G., Panara, R.B., 1990. Specifying software quality with metrics. In: Thayer, R.H., Dorfman, M. (Eds.), System and Software Requirements Engineering. IEEE Computer Science Press, Silver Spring.

Kekre, S., Krishnan, M.S., Srinivasan, K., 1995. Drivers of customer satisfaction for software products: implications for design and service support. Management Science 41 (9), 1456–1470.

Luce, D.R., Tukey, J.W., 1964. Simultaneous conjoint measurement: a new type of fundamental measurement. Journal of Mathematical Psychology 1 (February), 1–27.

McCabe, T., 1976. A complexity measure. IEEE Transactions on Software Engineering SE-2, 308–320.

Nunnally, J.C., 1978. Psychometric Theory. McGraw Hill, New York.

Perry, D.E., Romanovsky, A., Tripathi, A., 2000. Current trends in exception handling. IEEE Transactions on Software Engineering. 26 (9), 817–819.

QAI, 1989. Measurement of the customer's view of information systems quality. QAI Research Report #1. Quality Assurance Institute, Orlando.

Wittink, D.R., Krishnamurthi, L., Reibstein, D.J., 1990. The effect of differences in the number of attribute levels on conjoint results. Marketing Letters 1 (2), 113–129.

Wonnacott, T.H., Wonnacott, R.J., 1984. Introductory Statistics for Business and Economics, third ed. Wiley, Chichester.

Bonnie Brinton Anderson is a Ph.D. student at the H. John Heinz III School, Carnegie Mellon University. Her research interests include organizations and information systems.

Akhilesh Bajaj is Assistant Professor of Information Systems Management, at the H. John Heinz III School, Carnegie Mellon University. He holds a Ph.D. in MIS from the University of Arizona, an MBA from Cornell University and a B.Tech. from the Indian Institute of Technology, Bombay. His research interests include building large-scale information systems in organizations, usability engineering and the evaluation, adoption and usage of information systems by individuals and organizations.

Wilpen Gorr is Professor of Public Policy and Management Information Systems at the H. John Heinz III School, Carnegie Mellon University. His research interests include information systems, including geographic information systems, and decision support systems.