Effects of intra-group conflict on packaged software development team performance

Steve Sawyer

School of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802 USA, email: sawyer@ist.psu.edu

Abstract. *Data from 40 packaged software development teams are used to test a path model that relates three antecedents, the presence of intragroup conflict and the level of conflict management to software development team performance. Findings indicate that a combination of the team's characteristics, team member characteristics and existing levels of intragroup conflict accounts for nearly one-half of the variance between the best and worst-performing teams. Furthermore, the level of conflict management moderates the relationship between existing levels of intragroup conflict and performance. These results highlight both the complexity of the social processes of packaged software development and the value of this perspective for gaining insight on software development performance.*

Keywords: Conflict, conflict management, packaged software, path models, software development, software development team performance, teams

INTRODUCTION

Intra-group conflict among members of a software development team is seen as both inevitable and likely to lead to bad consequences (Brooks, 1974; Robey, 1984). There have been two general approaches to minimizing such conflict. In the software engineering literature the typical approach is to hire the best individuals (Boehm, 1981; Walz *et al*., 1993). The information systems development literature has emphasized conflict resolution and the selection of people who are socially skilled to do so. The primary focus of this second line of research has been to focus on the relationship between the team's members and various users/external stakeholders (Robey, 1984; Robey *et al*., 1989; 1993). Both literatures suggest that some combination of the 'right' skill blend (be they technical or social) will reduce the negative consequences of conflict. Thus, the contemporary advice to software development managers is to hire 'team players' and to carefully assemble teams of 'compatible' people.

We also believe that intragroup conflict is inevitable (Simmel, 1955). However, like others, we contend that conflict, if it is well managed, may improve the software development team's performance (Carmel & Sawyer, 1998; Zachary, 1998). From this perspective, the presence of conflict is less important than how it is managed: if a team manages their intragroup conflict well they should out-perform a team that does not, even if the relative levels of conflict are similar. This shifts the managerial emphasis from picking team members that can work without too much conflict to selecting team members who can work well in an open, and often contentious, social environment (McCarthy, 1995). The research reported on in this paper explores:

1 What factors most affect the level of intragroup conflict in packaged software development teams?
2 What effects do these factors have on packaged software development team performance?

To respond to these questions we draw on data collected from 40 packaged software development teams at one site of a large, global, computer software and hardware manufacturer. Selecting the teams from one site minimizes the potential confounding effects of varying industrial/organizational factors. That is, by selecting teams from one site, we know that they share common production goals, have a common software development methodology and use the same software development tools. Project managers for these teams are trained in the same techniques and also report to the same senior managers.

Studying packaged software development teams also allows us to explore the role of intragroup conflict in a different context. Packaged software [also known as commercial, commercial-off-the-shelf (COTS) or shrink-wrap] is licensed for use by others (Carmel, 1997). Carmel & Sawyer (1998) argue that packaged software development differs from custom development along a number of dimensions. The most pertinent of these dimensions for this paper is the level of user involvement. As Keil & Carmel (1995) show, there is little direct contact between most developers and users of packaged software. This means existing models regarding the effects on intragroup conflict and performance due to the involvement and/or participation of users (i.e. Robey *et al*., 1993) cannot be used without some additional conceptualizing.

The paper continues in five sections. In the next section, we develop the concepts of intragroup conflict among the members of packaged software development teams (called conflict through the remainder of the paper), theorize about the antecedents to conflict and detail what we mean by software development performance. The second section lays out how these constructs are measured. The research approach and data collection efforts are presented in the third section, while the fourth contains the analyses and findings. The paper concludes with a discussion of the findings and their implications to software development management and research.

CONFLICT AMONG SOFTWARE DEVELOPMENT TEAM MEMBERS

Intra-group conflict is a general social phenomenon and the relevant literature contains several definitions that encompass factors such as objective conditions, emotions, perceptions and behaviour (Simmel, 1955; Pondy, 1967; Deutsch, 1969; Thomas, 1975; Green & Taber, 1980; Wall, 1987; Volkema & Bergman, 1989). Rather than attempt to argue the superiority of any one specific definition, in this paper we treat conflict generically to be a difference between two or more people about the meaning of some information (such as a requirement or need, an idea, or a decision). Thus, the existence of conflict indicates neither a positive nor negative state, just a difference between how people interpret information (Simmel, 1955).

Conflict management (also known as conflict resolution) is the process of resolving these differences (Green & Taber, 1980; Sambarmurthy & Poole, 1992). The resolution of conflict can lead to both positive and negative outcomes (Simmel, 1955; Deutsch, 1969). For example, Robey's work on conflict among software developers posits conflict as a mostly negative effect (Robey, 1984; Robey & Farrow, 1989; Robey *et al*., 1989; 1993). Walz *et al*. (1993) presents evidence highlighting both the positive and negative aspects of conflict among software developers.

There are at least two reasons why conflict and its management are critical aspects of team-based software development. The first reason is that conflict is endemic among people when they work together (Pondy, 1967; Thomas, 1975; Green & Taber, 1980). For instance, when two developers bring contradictory data regarding system requirements to a meeting, the members of that software development team must sort through the differences and reach a shared understanding of the meaning of these disparate pieces (Crowston & Kammerer, 1998). Resolving this type of conflict demands both an articulation of differences and a negotiation of alternatives to develop a reasonable compromise, agreement, or shared understanding (Pondy, 1967; Thomas, 1975; Robey *et al*., 1989; Walz *et al*., 1993). Moreover, there is a diminished benefit for working together in teams if this does not create some level of conflict (Simmel, 1955). Group work provides one means of bringing multiple perspectives to bear on a common problem. A lack of conflict among team members is often called 'group-think' and the literature provides several examples of poor decisions from groups with *too little* conflict (Deutsch, 1969; Janis, 1982).

The second reason conflict is an important aspect of team-based software development is the relationship between conflict management and team-level performance (Robey *et al*., 1993; Walz *et al*., 1993; Barki & Hartwick, 1994). From the example above, constructive conflict management would use the two differing requirements needs to improve the shared understanding of the issues, leading to improved team efforts (Pondy, 1967; Thomas, 1975; Green & Taber, 1980; Robey *et al*., 1989; Walz *et al*., 1993). However, failing to resolve the differences between the two pieces of information the developers brought to the team is likely to have negative consequences. Should conflict be badly managed, and a consensus not reached, ill-feelings may fester, ambiguity over the requirements may increase and the ability to communicate openly may be inhibited (Pondy, 1967; Thomas, 1975; Green & Taber, 1980; Robey *et al*., 1989; Walz *et al*., 1993). Furthermore, patterns of poor conflict management encourage people to not contribute to the team's effort. This is the exact antithesis of working together (Simmel, 1955; Pondy, 1967; Thomas, 1975). For example, Curtis *et al*. (1988) find software developers: (1) often do not know enough about their system's operational domain, (2) are subjected to conflicting and ever-changing requirements and (3) have trouble communicating and co-ordinating their work. All three factors can raise the level of conflict in the development team.

STUDYING CONFLICT IN PACKAGED SOFTWARE DEVELOPMENT TEAMS

To date, the literature on the role of intragroup conflict and its effects in software development has focused primarily on conflicts between users and developers (a broad term used here to mean analysts, programmers and their managers) (Robey, 1984; Robey & Farrow, 1989; Robey *et al*., 1989; 1993; Barki & Hartwick, 1994). However, software development is increasingly being carried out by specialized firms. Thus, the interaction between users and developers is both diminishing and typically being carried out by intermediaries (Keil & Carmel, 1995; Carmel, 1997; Carmel & Sawyer, 1998). When the user is not a direct participant in software development, existing models of intragroup conflict in software development may be inadequate (see Robey *et al*., 1993; Barki & Hartwick, 1994).

For example, there is evidence that conflict *among* developers is at least as important to performance as is the conflict between developers and users (Walz *et al*., 1993). Zachary (1994; 1998) highlights contention as the single predominant means of interaction among packaged software developers. He argues that these teams are set up to enforce an 'armed truce.' In his study of Microsoft's developers he found that 'Consensus was not sought because it was not desired. Conceptual stalemates did not stymie activities. . .. The ethos of armed truce also forced advocates of a controversial decision to continually defend themselves' (p. 64). Carmel & Sawyer (1998) argue that the team dynamics of packaged software – where several members of a team may be either millionaires (or working for free in the hopes of becoming a millionaire) – are markedly different than the development work currently discussed in the literature. The increased focus on contention as a means of working together in packaged software development also implies that managing conflict is even more central to successful team performance.

MODELLING CONFLICT IN PACKAGED SOFTWARE DEVELOPMENT TEAMS

As shown in Figure 1, defined in Table 1, discussed below and detailed in Appendix A, we posit that the level of existing conflict in a packaged software development team is driven by four sets of antecedents: organizational characteristics, project characteristics, team characteristics and team member characteristics. In turn, the effect of the existing level of conflict on software development team performance is moderated by the extent that conflict is reduced: this is where the positive (and/or negative) effects of conflict management are realized.

Antecedents of conflict

At a broad level, Thomas (1975) argues that conflict stems from individual behavioural predispositions: the social pressures in working together and incentive structure, rules and/or procedures of the organization. The four sets of antecedents to conflict in our model build on Robey's (1984) categorization of conflict in software development. He posits four sources of conflict: individual differentiation, sharing of resources, interdependence and distribution of

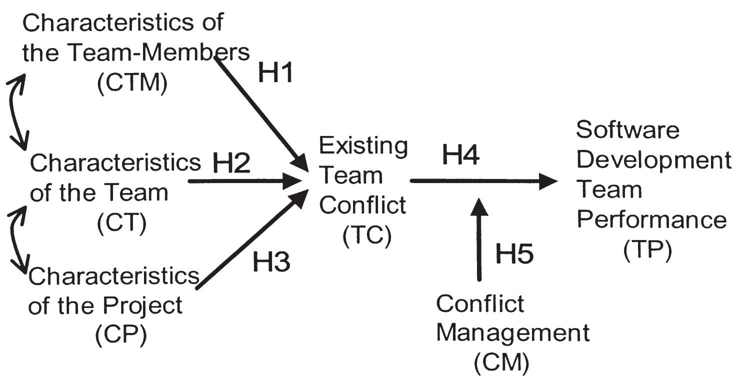

Characteristics of the Team-Members (CTM)
H1
Characteristics of the Team (CT)
H2
Characteristics of the Project (CP)
H3
Existing Team Conflict (TC)
H4
Software Development Team Performance (TP)
H5
Conflict Management (CM)

Figure 1. Conflict management model.

Table 1. Antecedents to conflict

Characteristics of the team members:	Individual behaviours Individual differentiation
Characteristics of the team:	Social pressures Sharing of resources Interdependence
Characteristics of the project:	Requirements volatility Resource availability
Characteristics of the organization*:	Organizational incentive structures rules and/or procedures Distribution of power

*Controlled for in this study by selecting teams from one site.

power. Differentiation refers to the specialization of roles which leads to differing interpretations across team members. Sharing reflects that resources are always scarce. Interdependence is a form of mutual need which demands co-ordination and negotiation. The distribution of power reflects both existing organizational and social pressures.

Individual characteristics are typically included in most models of conflict among software developers. For example, Robey *et al*. (1993) build on the work of Robey *et al*. (1989) to highlight how individual levels of participation and influence affect conflict, its resolution and project success. Barki & Hartwick (1994) test a similar model and highlight how individual disagreements further contribute.

Team-level characteristics serve as another common antecedent to conflict. For example, Newman & Robey (1992) highlight how the act of working together creates a set of social structures within which the software developers work and that guide resource allocation. Kiesler *et al*. (1994) describe how resource sharing affects the way software developers work together. Sawyer *et al*. (1997) describe how the level of interdependence between software developers on the same team shapes how they deal with conflict.

Project characteristics are also common antecedent of conflict among software developers. For example, at a broad scale, Kumar & Van Dissel, 1996) argue that resource availability is the primary reason behind collaboration. In software development both Curtis *et al*. (1988) and Walz *et al*. (1993) provide evidence showing that requirements volatility is an antecedent to conflict. Furthermore, the issues of resource availability are highlighted by how Microsoft often sets up its project teams to compete for resources (Zachary, 1994; 1998). Finally, as mentioned above, data are drawn from software development teams at one site. This provides some control over organizational characteristics. Thus, data regarding these variables are not collected from each team.

Conflict

In this paper we explore whether the existence of conflict is driven by certain antecedents (i.e. Thomas, 1975). Thus, to test this relationship we use a factor model of conflict to portray the relationship of factors that contribute to conflict (Pondy, 1967; Markus & Robey, 1988). Furthermore, our approach is to look at the factors, or structures, that help to account for variations in conflict. From a structural perspective there are three types of conflict in formal organizations: bargaining, bureaucratic and systematic (Pondy, 1967). Bargaining conflict occurs among parties who have an interest in maintaining and encouraging a shared relationship. For example, a bargaining conflict arises over how best to placate a disgruntled team member. Bureaucratic conflict occurs between two or more parties where some form of power relationship (superior–subordinate) exists. Systematic conflict occurs among parties as part of a lateral or working relationship. For example, the tensions which often arise between quality assurance people and software developers on the same team exist because they often have different and even divergent goals that are difficult to synthesize. Furthermore, each form of conflict can affect performance differently and may draw on antecedents in different ways (Simmel, 1955; Pondy, 1967; Thomas, 1975). However, in keeping with the broad perspective of conflict used in this paper, we acknowledge the existence of these forms, but treat them collectively.

Managing conflict

With any level of existing conflict, some effort is made to manage it (Deutsch, 1969). Examples of such efforts include the use of humour to minimize the tension among team members. Other efforts to manage conflict include attending to clear communication, co-ordinating work tasks and resolving differences as they arise. These actions suggest that conflict management serves as a moderating factor between the level of conflict and performance (Venkatraman, 1989) A moderating influence means that variations in the levels of the moderating variable (in this case conflict management) affect the relationship between two variables (in this case the level of existing conflict and software development team performance). That is, we posit that the level of conflict management affects the relationship between existing levels of conflict and software development team performance.

Conceptualizing conflict management as a moderating variable further implies two things. On the one hand, the total amount of conflict should be a less important predictor of software development team performance than is the *unresolved* conflict that remains after the team's efforts to manage conflict. So, high levels of existing conflict, well managed, can lead to small levels of unresolved conflict and superior performance. Conversely, poor conflict management may lead to relatively high levels of unresolved conflict even if there are low levels of existing conflict.

The other implication of this conceptualization of conflict management is that it does not *directly* affect software development performance. As a moderating factor, variations in the level of conflict management affect the relationship between existing levels of conflict and software development team performance and not the two factors. Treating conflict management as a moderating factor means it can serve as a magnifier. Constructive conflict management can improve performance and negative conflict management can reduce performance.

Software development team performance

Software development team performance demands multiple measures as no single metric can capture a total picture of the effort (Kemerer, 1989; Delone & McLean, 1992; Guinan *et al*., 1997). Many contemporary metrics (e.g. defect rates, function points, line-of-code, complexity metrics, elapsed time and resource consumption) typically focus on the production aspects of software development. However, production measures provide a limited picture with little insight on how the software is perceived and used by the user.

Other measures of software development team performance (such as perceived product quality and user satisfaction) tap less tangible, but no less crucial, aspects of software development (Bayer & Melone, 1989; Curtis, 1989; Delone & Mclean, 1992; Henderson & Lee, 1992). However, to be useful, perceptual measures should be assessed by people external to the development teams (stakeholders in this study) to avoid response bias (Seidler, 1974; Lee *et al*., 1991). A team's stakeholders are typically distant from the team's internal efforts and rely on specific quantitative data such as project reports and meeting budget/schedule commitments and also their perceptions of the team members competence to assess performance.

Combining objective and perceptual measures is the most robust way to measure software development performance (Kemerer, 1989). However, at this site, data on defect rates, product size, complexity and delivery (time) schedules are collected at the product level. The data cannot be directly linked to the individual project teams in this study as many teams work on one product. In this sample, from three to six teams worked together on a shared product. The teams in the sample worked on a total of eight distinct products. Thus, in this paper, we use perceptual measure to evaluate software development team performance. These perceptual measures include: the quality of the software, the ability of the team to work together effectively, the efficiency of the team and satisfaction with the resulting product. The perceptual performance measures are asked of stakeholders to avoid self-report bias.

HYPOTHESIZING A MODEL

To respond to the two research questions posed at the beginning of this paper, we test the path model depicted in Figure 1. We use path analytic techniques for two reasons. First, they provide a means to test a multistage model (Pedhazur & Schmelkin, 1991). Second, path models are often employed in behavioural analysis when a moderation model is posited (Venkatraman, 1989). In this section we explicitly hypothesize five paths (as shown in Figure 1) and implicitly hypothesize that all other paths are trivial. However, path analytic techniques demand assessing all paths to ascertain their contribution. Given this, hypotheses H1, H2 and H3 test the extent to which characteristics of the team members, the team and the project directly affects the existing levels of conflict among the team's members.

H1: Characteristics of the team members are significant predictors of existing levels of packaged software development team conflict.
H2: Characteristics of the team are significant predictors of existing levels of packaged software development team conflict.
H3: Characteristics of the project are significant predictors of existing levels of packaged software development team conflict.

Hypothesis H4 tests to see whether existing levels of conflict directly affect software development team performance.

H4: Existing levels of conflict will be a significant predictor of software development team performance.

Hypothesis H5 tests to see whether conflict management moderates the relationship between existing conflict and software development performance (Venkatraman, 1989). This moderating effect means, for example, that higher levels of conflict management should magnify the inverse relationship between lower levels of unresolved conflict and higher levels of team performance.

H5: Conflict management will moderate the relationship between existing levels of conflict and software development team performance.

RESEARCH APPROACH AND DATA COLLECTION

Our field-based study employed cross-sectional surveys of development teams coupled with phone-based surveys of the team's stakeholders. Data were collected using a key-informants approach (Campbell & Stanley, 1966; Seidler, 1974; Lee *et al*., 1991; Henderson & Lee, 1992). Key informants are members of the team selected to provide a broad sample of the views and perceptions of the entire team. Key informants for the teams included in this study included the project leader, a key technical lead and at least one project member. Both surveys were pre-tested and pilot tested at the site prior to use (Dillman, 1978). Respondents and stakeholders were promised anonymity.

The researcher contacted each team directly about being part of the study. Participation was voluntary but, to be included, each team had to agree to contribute at least three surveys. One motivation for participation was a follow-up meeting with each team to discuss the findings. Members of 50 teams volunteered to participate. Of these 50 teams, 46 teams provided data. Six teams (representing 14 respondents) which provided data were not software development teams and are not included in this analysis, leaving a sample of 40 teams.

The teams included in this study build software for commercial sale. These commercial software products are large, popular and include both databases and language compilers. Some of these products have been in the market for more than two decades. Each team in the sample is typically charged with one (or more) modules that, when combined, make up the overall product. What this means is that, while the modules may be distinct segments and identifiable at some level, they are also tightly integrated together. In development, this means that teams are often highly dependent on one another and that the dependencies are not sequential as two modules may pass data back and forth during the run which means the two teams work closely together as both customer and producer.

Team size ranged from five to 15 people, with most less than 10, and they have been organized in a team-based manner for several years. Team members were surveyed as they completed their most recent module/project. The 40 teams participating in this study, while not randomized, represent approximately 50% of the project teams at this site and include teams from all departments. The 128 respondents represent nearly 11% of all developers. The lower number of individual respondents is an artefact of the key informants approach as we asked only a subset of each team to complete a survey.

For each team surveyed stakeholders were contacted to provide an external evaluation of team and product performance. As indicated, stakeholders are people who are affected by the team, or work with the team but not on a daily basis. Examples are senior managers, user managers and consultants. Teams identified their stakeholders, who were then contacted by the researchers to ask for participation. Fifty-six stakeholders (a 75% response rate) agreed. Each team had at least one stakeholder. Each stakeholder had detailed knowledge of their team and of the overall product, but only limited knowledge of other teams involved with that product.

In Table 2 we present the reliability statistics for the factors used in this study. A coefficient of 0.70 is considered satisfactory, and 0.90 or higher is excellent, as evaluated using Chronbach's (1951) alpha test. The indicators are provided in Appendix A.

In Table 3 we present some of the pertinet sample demographics. These show that both developers and stakeholders are well educated and have extensive software development experience. The teams have an average of 7.1 years of software development experience per member. However, while the team membership is relatively stable, working together for an average of nearly 2 years, they change leaders almost yearly.

In this study, the team is the level of theory, measurement and analysis (Klein, Danserou & Hall, 1994). This means that the survey and interview questions were asked with regard to team-level behaviours and data are analysed at the team level so individual responses have been aggregated by the team (Jones & James, 1979; James, 1982). Table 4 presents the

Table 2. Factor reliabilities

Factor (number of indicators)	Reliability (Chronbach alpha)
CP: characteristics of the project (4)	0.70
CT: characteristics of the team (5)	0.72
CTM: characteristics of the team members (4)	0.77
TC: team (existing) conflict (4)	0.8
CM: conflict management (3)	0.70
TP: stakeholder-rated team performance (9)	0.93

Table 3. Sample demographics

Factor	Respondents (mean)	Stakeholders (mean)
Software development experience	7.25 years	13.9 years
Time with company	4 years	10.1 years
Time with present team	1.9 years	2 years
Number of previous projects	9.8	31
Education:	89% have a BA/BS and 41% of these also have MS/PhD.	

Site restrictions prevented us from collecting gender and age data.

Table 4. Means, standard deviations and correlations between factors (n = 40)

Factor	Mean (SD)	CP	CT	CTM	TC	CM
Project characteristics (CP)	4.37 (0.94)					
Team characteristics (CT)	4.12 (0.91)	0.564*				
Team-member characteristics (CTM)	4.32 (0.93)	0.594*	0.681*			
Team (existing) conflict (TC)	5.26 (1.10)	0.417*	0.458*	0.296		
Conflict management (CM)	4.46 (1.13)	0.212	0.637*	0.453*	0.382†	
Stakeholder-rated team performance (TP)	5.07 (0.72)	0.201	0.484*	0.340†	0.402*	0.260

† $P < 0.05$, * $P < 0.01$.
Seven-point scale with 1 = low/poor/worst and 7 = high/excellent/best.

mean, standard deviation and correlation for each factor used in the analysis. The significant correlations between the antecedent variables suggests that there may be low discriminant validity and raises concerns regarding multicollinearity (which will be discussed again in the next section). As discriminant validity criteria are debated and criterion not well formed (see Pedhazur & Schmelkin, 1991, p. 75), Campbell and Fiske's multitrait/multimethod approach is often used to help assess discriminant validity (see Campbell & Stanley, 1966). Given the uni-method approach taken, this comparison is not possible. Instead, the correlations of each indicator for each variable were inspected to see whether the within-construct correlations were higher and more significant than the between-variable indicator correlations. Of the 13 indicators used to develop the three antecedent variables, only two indicators correlated more strongly across variables than within the variable. Thus, we retain the three variables on the basis of their conceptual relevance.

ANALYSIS AND FINDINGS

To assess the multistage model presented in Figure 1, we use path analysis. Path analytic techniques provide a means to decompose the total effects of one variable on another into direct, indirect and spurious elements. The total effects are the zero-order correlations and the direct and indirect effects are calculated using the standardized coefficients of multiple regressions. These are computed from the path relationships hypothesized and presented in Figure 1 (Duncan, 1966; Spaeth, 1975).

Path analysis is premised on a fully specified model so all paths must be calculated (Duncan, 1966; Spaeth, 1975). However, as any argument for causality must be conceptually based (as path models only imply a preconceived causal ordering) trivial paths are trimmed from the analysis and the model re-estimated (Spaeth, 1975; Pedhazur & Schmelkin, 1991). The determination of what is a trivial path remains a topic of debate (see Duncan, 1966; Spaeth, 1975; Cohen & Cohen, 1983; Pedhazur & Schmelkin, 1991). Typically, direct path effects of 0.10 or less can be trimmed. One of the side-effects of trimming paths is that this increases the amount of spurious – or unmeasured – effects as the indirect components of the trimmed variables cannot be estimated. This means trimming also reduces the descriptive power of the model and leads to a more conservative estimation.

Because path analysis draws on multiple regression and correlation (MRC) techniques, multicollinearity may be an issue (Spector, 1977). The potential effects of multicollinearity are assessed by calculating each variable's tolerance (Pedhazur & Schmelkin, 1991, p. 436). Tolerance refers to the portion a given variable does not share with all other independent variables, and can range from 0.0 to 1.0 (where 1.0 means total independence). For a model with two independent variables, tolerance can be assessed by the correlation between the variables. For models with three or more independent variables tolerance must be calculated separately and is typically an option in most statistical packages such as SPSS (which was used to support this analysis). Tolerance statistics are reported in Appendix B.

Table 5 summarizes the regression analyses used to build the path model(s) testing the first four hypotheses. The table includes the direct, indirect and spurious effects for each path in both the fully specified and trimmed models. The trimmed model, with all direct paths of less than 0.10 removed, is also shown graphically in Figure 2. The trimmed model has two new paths that indicate a direct effect from antecedents to performance. Furthermore, several hypothesized links are not found to be significant.

Table 6 provides correlations for the split-sample analysis and Tables 7 and 8 summarize the analyses testing hypothesis H5. Because H5 is a moderation hypothesis, the 40-team sample is divided in two based on the level of the moderating variable (conflict management). The data in Table 6 show that the existing conflict among the two groups are non-significantly different. However, the correlations among these variables are distinctly

Table 5. Regression analysis supporting full and trimmed path models (H1–H4)

Independent variable (standardized betas)						Dependent		
Model	Int.	CP	CT	CTM	TC	Variable	Adj. r^2	F(*P*)
Full	–0.467	0.101	0.406*	0.284	0.106	TP	0.442	9.13 (.000)
Trimmed	–0.297	removed	0.451**	0.289*	0.127	TP	0.450	12.17 (.000)

* $P \leq 0.05$; ** $P \leq 0.01$.

Full model

Effects	Total	Direct	Indirect	Spurious
CP→TC	0.416	0.042	0.320	0.054
CT→TC	*0.457*	*0.186*	*0.094*	*0.178*
CTM→TC	0.206	0.059	0.000	0.147
CP→TP	0.600	0.061	0.682	–0.142
CT→TP	*0.641*	*0.260*	*0.417*	*–0.036*
CTM→TP	*0.521*	*0.148*	*0.081*	*0.292*
CT→TP	0.393	0.042	0.000	0.351

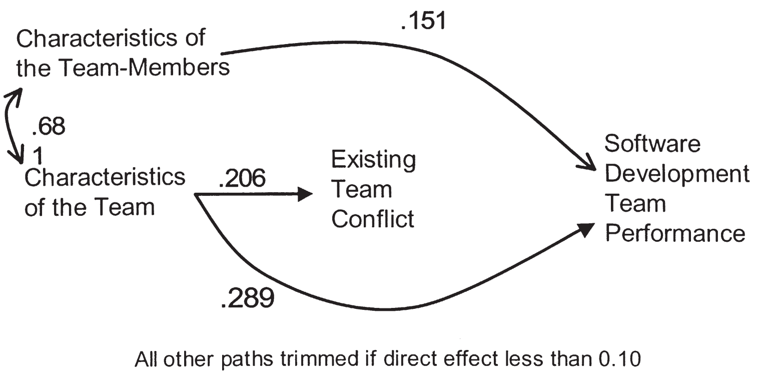
Trimmed model

Effects	Total	Direct	Indirect	Spurious
CT→TC	*0.457*	*0.206*	*0.094*	*0.157*
CTM→TC	0.206	0.060	0.000	0.146
CT→TP	0.641	0.289	0.417	–0.065
CTM →TP	0.521	0.151	0.081	0.289
CT→TP	0.393	0.050	0.000	0.343

different. For example, the correlations between antecedents and levels of existing conflict differ. Figure 3 provides a graphical representation of the teams with low levels of conflict management. Figure 4 provides the same representation for the team with high levels of conflict management.

The two subsample models are similar to each other and to the full sample's model, though the weights of the effects differ. And, in the low conflict management model, there is a direct effect of conflict on performance which is not present in the high conflict management model. This difference between the two models is evidence of the moderating effect played by conflict management. Furthermore, in Figure 4, the antecedents are no longer significantly correlated. This is additional evidence of a difference in the ways these two groups of teams manage conflict.

The aggregate performance measure of the two subsamples is not significantly different. However, further investigation of the performance differences provides more evidence. If the performance data of the entire sample is divided into four subsamples and the bottom (worst



Characteristics of the Team-Members
.151
.68
1
Characteristics of the Team
.206
Existing Team Conflict
Software Development Team Performance
.289
All other paths trimmed if direct effect less than 0.10

Figure 2. Conflict management path model.

Table 6. Means, standard deviations and correlations between factors used in split-sample analysis

Teams with lower levels of conflict management (n = 22‡)

Factor	Mean (SD)	CP	CT	CTM	TC
Project characteristics (CP)	4.24 (0.66)				
Team characteristics (CT)	3.80 (0.78)	0.499*			
Team member characteristics (CTM)	3.68 (1.03)	0.520*	0.559*		
Team (existing) conflict (TC)	5.07 (1.03)	0.599*	0.469*	0.196	
Stakeholder-rated team performance (TP)	4.65 (1.15)	0.400*	0.615*	0.545*	0.444†

Teams with higher levels of conflict management (n = 18‡)

Factor	Mean (SD)	CP	CT	CTM	TC
Project characteristics (CP)	4.57 (1.28)				
Team characteristics (CT)	4.64 (0.87)	0.633*			
Team-member characteristics (CTM)	6.63 (0.77)	0.013	–0.154		
Team (existing) conflict (TC)	5.56 (0.88)	0.259	0.319	–0.096	
Stakeholder-rated team performance (TP)	5.55 (1.18)	0.508*	0.609*	0.335	0.185

† $P < 0.05$, * $P < 0.01$.

Seven-point scale with 1 = low/poor/worst and 7 = high/excellent/best.

‡ Split-sample sizes differ because several teams were grouped together as their values were identical.

performing) quartile is compared with the top (best performing quartile), two insights arise. First, all but one of the worst performing teams have low levels of conflict management whereas all but one of the highest performing teams have high levels of conflict management. Second, a comparison of these two quartile's performance values, using the Wilcox rank-ordering test for non-parametric data (Howell, 1987, p. 558), shows a significant difference. It is possible that the small sample size confounds parts of this analysis.

Table 7. Regression analysis supporting full and trimmed path models: for lower levels of conflict management

Independent variable (standardized betas)						Dependent		
Model	Int.	CP	CT	CTM	TC	Variable	Adj. r^2	F(P)
Full	0.131	–0.137	0.339	0.369	0.295	TP	0.388	4.96 (0.006)
Trim	–0.296	removed	0.334	0.314	0.226	TP	0.405	6.68 (0.002)

* $P \le 0.05$; ** $P \le 0.01$

Lower CM model, full

Effects:	Total	Direct	Indirect	Spurious
CP→TC	0.599	–0.082	0.336	0.345
CT→TC	0.469	0.159	0.110	0.200
CTM→TC	0.196	0.072	0.000	0.124
CP→TP	0.400	–0.055	0.856	–0.401
CT→TP	0.615	0.208	0.513	–0.106
CTM→TP	0.545	0.201	0.087	0.257
TC→TP	0.444	0.131	0.000	0.313

Lower CM model, trimmed

Effects:	Total	Direct	Indirect	Spurious
CT→TC	0.469	0.157	0.110	0.203
CTM→TC	0.196	0.062	0.000	0.134
CT→TP	0.615	0.205	0.513	–0.103
CTM→TP	0.545	0.171	0.087	0.287
TC→TP	0.444	0.100	0.000	0.344

CONFLICT IN PACKAGED SOFTWARE DEVELOPMENT TEAMS

Findings indicate that a revision to the hypothesized model presented in Figure 1 can account for nearly one-half of the variance in software development team performance among the 40 teams in this sample. That is, the analyses presented here provide support for H2 and H4. And, as hypothesized (H5), variations in the level of conflict management moderates the relationship between existing levels of team conflict and team performance. This moderation is seen in the differences between high and low levels of conflict management. In the teams with low levels of conflict management there is a significant link between existing levels of conflict and performance (see Figure 3). This is not so for teams with higher levels of conflict management (see Figure 4).

The trimmed models presented in Tables 7 and 8 have significantly different intercepts (based on a Wilcox rank-ordered test, Howell, 1987, p. 558). This was tested by developing a jack-knife sample. There are several means to test whether there are significant differences

Table 8. Regression analysis supporting full and trimmed path models: for higher levels of conflict management

Independent variable (standardized betas)						Dependent		
Model	Int.	CP	CT	CTM	TC	Variable	Adj. r^2	F(*P*)
Full	–2.781	0.125	0.594*	0.425*	0.003	TP	0.412	4.63 (.040)
Trim	–2.916	removed	0.673**	0.440*	0.012	TP	0.450	5.08 (.017)

$^*P \leq 0.05$; $^{**}P \leq 0.01$.

Higher CM model, full

Effects:	Total	Direct	Indirect	Spurious
CP→TC	0.259	0.032	0.201	0.026
CT→TC	0.319	0.189	0.015	0.115
CTM→TC	–0.096	–0.041	0.000	–0.055
CP→TP	0.508	0.064	0.438	0.007
CT→TP	0.609	0.362	0.007	0.240
CTM→TP	0.335	0.142	–0.018	0.210
TC→TP	0.185	0.001	0.000	0.184

Higher CM model, trimmed

Effects:	Total	Direct	Indirect	Spurious
CT→TC	0.319	0.215	0.015	0.090
CTM→TC	–0.096	–0.042	0.000	–0.054
CT→TP	0.609	0.410	0.007	0.192
CTM→TP	0.335	0.147	–0.018	0.205
TC→TP	0.185	0.002	0.000	0.183

between two regression models. Two possible empirical tests are the difference in r^2 and the difference in the intercept. Given the small sample, all tests must be non-parametric. Wilcox's rank-order test is used because it is a distribution free test (Howell, 1987, p. 558). Using Wilcox's rank-ordering requires a jack-knife sample of models. That is, for each of the two subsamples one team's data were systematically removed from the group and a model was developed. This led to two sets of models (of $n = 18$ and $n = 22$). Each has an intercept and adjusted r^2. These were ranked and the ranks summed by group to allow for rank-order testing. This led to a non-significant difference for the two sample's adjusted r^2. The two sample's intercepts are significantly different. Moving from empirical tests of difference to a conceptual basis, the differences in the path models suggests that there are structural differences in how conflict is linked to performance in these software development teams.

Hypothesis H1 is not supported. The hypothesized effect of team member characteristics on existing levels of team conflict is not significant. However, there is an unhypothesized direct effect of team member characteristics on performance. Finally, and contrary to the hypothe-

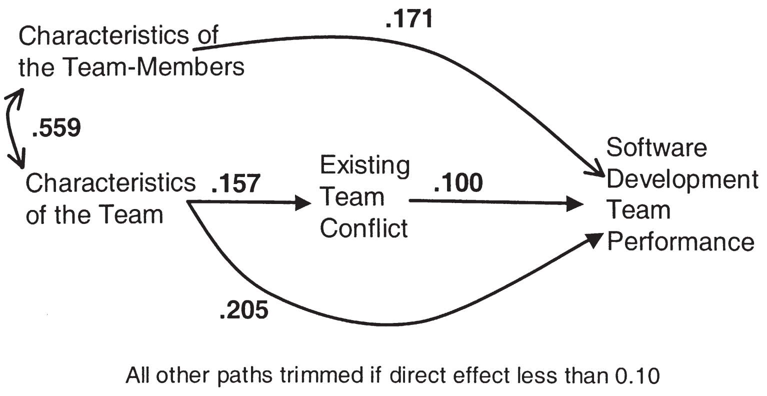

Characteristics of the Team-Members
.171
.559
Characteristics of the Team
.157
Existing Team Conflict
.100
Software Development Team Performance
.205
All other paths trimmed if direct effect less than 0.10

Figure 3. Low conflict management path model.

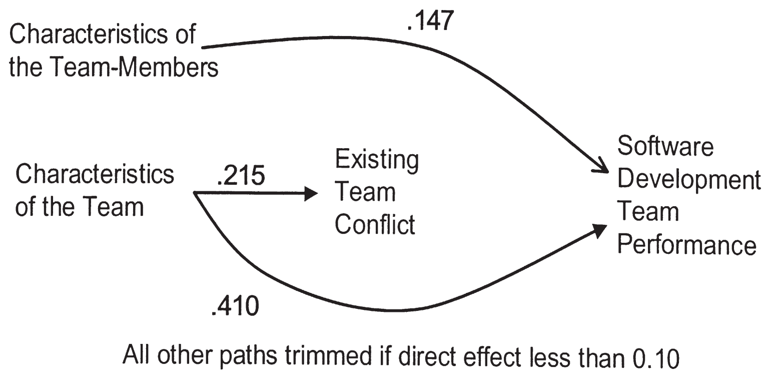

Characteristics of the Team-Members
.147
Characteristics of the Team
.215
Existing Team Conflict
Software Development Team Performance
.410
All other paths trimmed if direct effect less than 0.10

Figure 4. High conflict management path model.

sized relationship in H3, the characteristics of the project provide no significant contribution to existing levels of team conflict (or to performance). Figure 5 summarizes the findings graphically by representing a composite of the moderated path analysis.

THE ROLES OF THE ANTECEDENTS IN CONFLICT

The characteristics of the project have no significant contribution to either existing levels of conflict or team performance. This finding suggests that the volatility of requirements and the lack of resources may not have as quantifiable and deleterious effect to software development team performance as is often argued (i.e. Curtis *et al*., 1988).

The characteristics of the team members provide some unexpected predictive value. Data show that team member characteristics have no direct effect on existing levels of conflict.

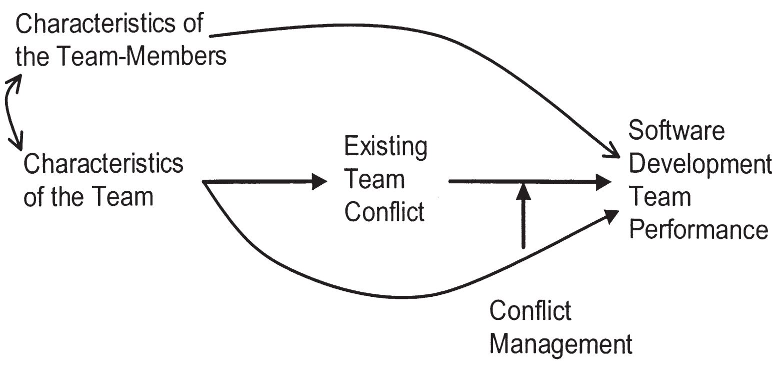

Characteristics of the Team-Members
Characteristics of the Team
Existing Team Conflict
Software Development Team Performance
Conflict Management

Figure 5. Revised conflict management model.

However, team member characteristics have an unhypothesized and direct effect on performance. Based on this analysis, the value of teamwork among software developers is slightly more important a predictor of performance than are individual contributions. This challenges the notion that good people will find a way to work together. The implicit message in Boehm's (1981) COCOMO estimators (which assesses solely the individual capabilities of a team) is that a team is equal to the sum of its parts. Data from this analysis suggest that is not so, the team can be greater than its parts.

The characteristics of the team are a significant predictor of both the level of existing conflict and team performance. This antecedent has both significant direct and indirect effects on team performance. This provides some additional evidence to support the contention that teamwork is critical to software development (Guinan *et al*., 1998; Sawyer & Guinan, 1998). Furthermore, it also provides some additional, albeit indirect, support for the premises of this paper. That is, managing conflict, not its presence or absence, is important to software development performance.

IMPLICATIONS FOR MANAGING SOFTWARE DEVELOPMENT TEAMS

Given the limitations of this study and acknowledging that this is exploratory work in the sense that the relationships between conflict and software development team performance need more attention, beyond this study, results suggest that improving intra-team conflict management can improve software development team performance. Furthermore, the evidence presented suggests that the ability to improve conflict management can stand independent of the existing levels of conflict.

As we have focused on packaged software development, these findings may be a result of domain differences. That is, the role of conflict may be different in the packaged environment than in the more often-studied custom IS arena (Carmel & Sawyer, 1998; Zachary, 1998; Sawyer, 2000). Conversely, it may be that the roles of conflict and its management transcend the domain differences of the artefacts being constructed. Certainly the possibility of domain differences demands additional research attention.

The findings of this study suggest two keys to improving packaged software development team performance. The first key is to more explicitly focus on training software developers to better manage the conflict endemic to working together. By this we mean that software developers should see conflict as both inevitable and potentially positive (e.g. Walz *et al*., 1993). And, packaged developers should also be trained to deal with their intra-team conflict. To avoid dealing with conflict will adversely affect software development team performance.

One means to achieve this conflict management learning is to tie such learning in with ongoing programs. For instance, training software developers on the means to manage conflict, often embedded in total quality management (TQM) programs (Zultner, 1993), can also be explicitly linked to the social processes of software development. In part, TQM programs provide a forum for discussion and encourage people to focus on the issues, to listen and to work together. A second potential vehicle to improve software developers' abilities to manage conflict is to rely more on existing technologies such as electronic meeting systems to either serve as a mediator between developers (i.e. Sawyer *et al*., 1997) and/or to provide chaperoned facilitation (i.e. Douglas *et al*., 1998). A third approach would be to rely on either informal training or self-education programmes. Currently, this is the primary means for helping to improve software developers' abilities in managing intra-team conflict, but it may not be the best way.

The second key to improving packaged software development team performance is to better account for how people deal with conflict in the methods used to produce software. For example, software development methods should explicitly provide for periods of differentiation and consensus building (Curtis, 1989; Walz *et al*., 1993). The importance of paying attention to the broader social milieu is evident in the methods such as socio-technical (Bostrom & Heinen, 1978a,b), soft-systems (Checkland & Scholes, 1990), Multiview (Avison *et al*., 1998) and ETHIC (Mumford, 1983). However, these approaches typically focus on the user–developer relationship and not the developer–developer relationship. Developer interaction-focused models of software production were first suggested about 10 years ago (Stefik *et al*., 1987; Curtis, 1989) and only recently has there been any discussion of such approaches. For example, Vessey & Sravanapudi (1995) argue that CASE tools can be used collaboratively. And Douglas *et al*. (1998) suggest the use of EMS for collaborative software development.

IMPLICATIONS FOR RESEARCH ON CONFLICT IN SOFTWARE DEVELOPMENT TEAMS

At least three insights for research arise from comparing the results of the full and split-sample analyses (see Figure 5). The first insight is that a model focused on the issues with conflict among software developers can account for a substantial portion of the variance between the best and worst performing teams in this sample. The second insight is that conflict management moderates the link between existing levels of conflict and performance. The third insight is the structural differences between teams with higher and lower levels of conflict management. For instance, the path models for the teams with lower levels of conflict management account for less variation. And, there is only a minimal direct link between existing conflict and team performance. These path models also highlight that there are large *indirect* effects associated with both team member and team characteristics.

The link between existing levels of conflict and team performance is trivial in the path model of teams with higher levels of conflict management. However, this model accounts for slightly more of the variance in team performance and the other major difference from the other split sample is the direct effect of team characteristics on team performance. This suggests that teams with higher levels of conflict management may rely on other aspects of their team's social processes to mitigate conflict. Team members' conflict management and characteristics may not be separated by existing levels of team conflict. This relationship demands additional exploration to better understand the causal linkage between conflict management and teams that have developed effective co-ordination mechanisms.

There are also several methodological constraints as data in this study data are collected using surveys. Both the questions and the way they are posed are critical determinants (Dillman, 1978; Pedhazur & Schmelkin, 1991). To maximize their value, the surveys drew on scales used in previous research (Green & Taber, 1980; Henderson & Lee, 1992). Still, the immature status of the scales is reflected in the minimally acceptable reliabilities for many of the factors. This is one reason why a relatively small percentage of the variance in software development team performance is explained. This suggests a need for more refined methods to explore the effects of conflict among software developers. For example, both Elam *et al*. (1991) and Robey (1994) suggest that only process models will provide the level of insight needed. The process-oriented work by Walz *et al*. (1993) and Dube (1998) support this assertion. As Newman & Robey (1992) discuss, process and factor models are complementary as they can each inform the other. The findings of this study suggest that a companion process-oriented study would be invaluable.

The use of perceptual data as the basis for performance may also be a concern. However, perceptually based performance data are often the best source as they are based on the perceptions of people who rely on the software for their work (Curtis, 1989; Henderson & Lee, 1992). Furthermore, the use of separate surveys to gather data from the developers and the stakeholders reduces the method bias due to having all factors on one survey (Pedhazur & Schmelkin, 1991; Dillman, 1978).

As the field continues to develop richer theories of intragroup conflict, the findings from this study suggest that one fruitful next step would be to test a broader sample of software developers and assess the industry/organization-level issues. In this study, the 40 team sample is not random, not large and comes from one site. These software development teams also build commercial, or packaged, software and this effort is markedly different from traditional internal or custom efforts (Carmel & Sawyer, 1998). Selecting a single site, however, was carried out explicitly to reduce potential confounds. A second goal was to gather data from actual software developers working in real teams on real projects. The potential loss of generalizability is the trade-off (Pedhazur & Schmelkin, 1991).

Given these concerns, the results of this study suggest that, for software development teams, how people work together is a stronger predictor of performance than the individual skills and abilities of the team's members. Because software development is carried out in an environment characterized by ambiguity, contradictory information and time pressures, effectively confronting conflict and thrive in a potentially contentious environment demands additional attention (Zachary, 1998). The data from this study shows that focusing on ways to effectively manage the endemic conflict between software development team members can account for nearly one-half of the variance between the most successful and least successful software development teams.

ACKNOWLEDGEMENTS

Partial funding for the study reported on in this paper was provided by IBM contract STL 92-543 and by a research grant from Syracuse University's Vice President for Research and Computing. The manuscript was substantially improved as a result of comments from Kevin Crowston, Bob Heckman, Ping Zhang, the associate editor and two anonymous reviewers. Particular thanks to Chuck Jiang for his work on earlier drafts of the paper.

REFERENCES

Avison, D., Wood-Harper, A., Vidgen, R. & Wood, J. (1998) A further exploration into information systems development: the evolution of Multiview 2. *Information Technology & People*, **11**, 140–151.

Barki, H. & Hartwick, J. (1994) User participation, conflict, and conflict resolution: The mediating role of influence. *Information Systems Research*, **4**, 422–438.

Bayer, J. & Melone, N. (1989) A critique of diffusion theory as a managerial framework for understanding adoption of software engineering innovations. *Journal of System Software*, **9**, 161–166.

Boehm, B. (1981) *Software Engineering Economics*. Prentice Hall, New York.

Bostrom, R. & Heinen, S. (1978a) MIS problems and failures: a socio-technical perspective. Part 1: The causes. *MIS Quarterly*, **1**, 17–32.

Bostrom, R. & Heinen, S. (1978b) MIS problems and failures: a socio-technical perspective. Part II: The application of socio-technical theory. *MIS Quarterly*, **1**, 11–27.

Brooks, F. (1974) The mythical man-month. *Datamation*, **7**, 44–52.

Campbell, D. & Stanley, J. (1966) *Experimental and Quasi-Experimental Designs for Research*. Rand-McNally, Chicago.

Carmel, E. (1997) American hegemony in packaged software trade and the 'culture of software'. *Information Society*, **13**, 125–142.

Carmel, E. & Sawyer, S. (1998) Packaged software development teams: what makes them different? *Information Technology & People*, **11**, 7–19.

Checkland, P. & Scholes, J. (1990) *Soft Systems Methodology in Action*. John Wiley & Sons, New York.

Chronbach, L. (1951) Coefficient alpha and the internal structure of tests. *Psychometrika*, **16**, 297–334.

Cohen, J. & Cohen, P. (1983) *Applied Multiple Regression / Correlation for the Behavioral Sciences*. Lawrence Erlbaum Associated, Hillsdale, NJ.

Crowston, K. & Kammerer, E. (1998) Collective mind in software requirements development. *IBM Systems Journal*, **36**, 1–24.

Curtis, W. (1989) Three problems overcome with behavioral models of the software development process. In: *Proceedings of the 11th International Conference on Software Engineering*, pp. 398–399. ACM Press, New York.

Curtis, W., Krasner, H. & Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, **31**, 1268–1287.

Delone, W. & McLean, E. (1992) Information systems success: the quest for the dependent variable. *Information Systems Research*, **3**, 60–95.

Deutsch, M. (1969) Conflicts: productive and destructive. *Journal of Social Issues*, **25**, 7–41.

Dillman, D. (1978) *Mail and Telephone Surveys: the Total Design Method*. John Wiley and Sons, New York.

Douglas, D., Lee, J., Pendergast, M., Hickey, A. & Nunamaker, J. (1998) Enabling the effective involvement of multiple users: methods and tools for collaborative software. *Journal of MIS*, **14**, 179–222.

Dube, L. (1998) Teams in packaged software development: the software corp. experience. *Information Technology & People*, **11**, 36–61.

Duncan, O. (1966) Path analysis: sociological examples. *American Journal of Sociology*, **72**, 1–16.

Elam, J., Walz, D., Curtis, B. & Krasner, H. (1991) Measuring group process in software design teams. In: *Information Systems Research: Contemporary Approaches and Emergent Traditions,* Nissen, H., Klein, H. & Hirschheim, R. (eds), pp. 51–61. North Holland, Amsterdam.

Green, S. & Taber, T. (1980) The effects of three social decision schemes in decision group performance. *Organizational Behavior and Human Performance*, **25**, 97–106.

Guinan, P., Cooprider, J. & Sawyer, S. (1997) The Effective Use of Automated Application Development Tools. *IBM Systems Journal*, **36**, 124–139.

Guinan, P.J., Cooprider, J. & Faraj, S. (1998) Enabling software development team performance during requirements gathering: a behavioral versus technical approach. *Information Systems Research*, **9**, 101–125.

Henderson, J. & Lee, S. (1992) Managing I/S Design Teams: a Control Theories Perspective. *Management Science*, **38**, 757–777.

Howell, D. (1987) *Statistical Methods for Psychology*, 2nd edn. Duxbury Press, Boston.

James, L. (1982) Aggregation bias in estimates of perceptual agreement. *Journal of Applied Psychology*, **67**, 219–229.

Janis, I. (1982) *Groupthink: Psychological Studies of Policy Decisions and Fiascos*. Houghton Mifflin, Boston.

Jones, A. & James, L. (1979) Psychological climate: dimensions and relationships of individual and aggregated work environment perception. *Organizational Behavior and Human Behavior*, **23**, 201–250.

Keil, M. & Carmel, E. (1995) Customer-developer links in software development. *Communications of the ACM*, **38**, 33–44.

Kemerer, C. (1989) An agenda for research in the managerial evaluation of computer-aided software engineering (CASE) tool impacts. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*. Hawaii, pp. 219–228.

Kiesler, S., Douglas, W., & Carley, K. M. (1994) Coordination as linkage: the case of software development teams. In: *Organizational Linkages: Understanding the Productivity Paradox*, Douglas. H. H. (ed.), pp. 214–239. National Academy Press, Washington, D.C.

Klein, K., Danserou, F. & Hall, R. (1994) Levels issues in theory development, data collection, and analysis. *Academy of Management Review*, **19**, 195–229.

Kumar, K. & Van Dissel, H. (1996) Sustainable collaboration: managing conflict and cooperation in interorganizational systems. *MIS Quarterly*, **20**, 279–300.

Lee, S., Goldstein, D. & Guinan, P. (1991) Informant bias in information systems design team research. In: *Information Systems Research: Contemporary Approaches and Emergent Traditions*, Nissen, H., Klein, H. & Hirschheim, R. (eds), pp. 635–656. North Holland, Amsterdam.

Markus, M. & Robey, D. (1988) Information Technology and Organizational Change: Conceptions of Causality in Theory and Research. *Management Science*, **34**, 583–598.

McCarthy, J. (1995) *Dynamics of Software Development*. Microsoft Press, Redmond, WA.

Mumford, E. (1983) *Designing Human Systems for New Technology: the ETHICS Method*. Manchester Business School, Manchester.

Newman, M. & Robey, D. (1992) A social process model of user-analyst relationships. *MIS Quarterly*, **16**, 249–266.

Pedhazur, E. & Schmelkin, L. (1991) *Measurement, Design and Analysis*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Pondy, L. (1967) Organizational conflict: concepts and models. *Administrative Science Quarterly*, **12**, 296–320.

Robey, D. (1984) Conflict models for implementation research. In: *Applications of Management Science*, R., Schultz & Ginzberg, M. (eds), pp. 89–105. JAI Press, Greenwich, CT.

Robey, D. (1994) Modeling interpersonal processes during system development: further thoughts and suggestions. *Information Systems Research*, **5**, 439–445.

Robey, D. & Farrow, D. (1989) User involvement in information system development: a conflict model and empirical Test. *Management Science*, **28**, 73–85.

Robey, D., Farrow, D. & Franz, C. (1989) Group process and conflict in systems development. *Management Science*, **35**, 1172–1191.

Robey, D., Smith, L. & Vijayasarthy, L. (1993) Perceptions of conflict and success in information systems development projects. *Journal of Management Information Systems*, **10**, 125–139.

Sambarmurthy, V. & Poole, M. (1992) The effects of variations in capabilities of GDSS designs on management of cognitive conflict in groups. *Information Systems Research*, **3**, 224–251.

Sawyer, S. (2000) Packaged software: implications of the differences from custom approaches to software development. *European Journal of Information Systems*, **9**, 47–58.

Sawyer, S. & Guinan, P. (1998) Software development: processes and performance. *IBM Systems Journal*, **37**, 120–144.

Sawyer, S., Farber, J. & Spillers, R. (1997) Supporting the social processes of software development teams. *Information Technology & People*, **10**, 46–62.

Seidler, J. (1974) On using informants: a technique for collecting quantitative data and controlling for measurement error in organizational analysis. *American Sociological Review*, **39**, 816–831.

Simmel. (1955) *Conflict and the Web of Group Affiliations*. The Free Press, New York.

Spaeth, J. (1975) Path analysis. In: *Introductory Multivariate Analysis*, Amick, D. & Walberg, H. (eds), pp. 162–179. MrCutchan, Berkeley.

Spector, P. (1977) What to do with significant multivariate effects in multivariate analysis of variance. *Journal of Applied Psychology*, **62**, 158–163.

Stefik, M., Foster, G., Bobrow, D., Kahn, K., Lanning, S. & Suchman, L. (1987) Beyond the chalk-board: computer support for collaboration and problem solving in meetings. *Communications of the ACM*, **30**, 32–47.

Thomas, K. (1975) Conflict and conflict management. In: *Handbook of Industrial Psychology*, Dunnette, W. (ed.), pp. 111–132. Rand McNally, Chicago.

Venkatraman, N. (1989) The concept of fit in strategy research: toward verbal and statistical correspondence. *Academy of Management Review*, **14**, 423–444.

Vessey, I. & Sravanapudi, P. (1995) CASE tools as collaborative support technologies. *Communications of the ACM*, **38**, 83–95.

Volkema, R. & Bergmann, T. (1989) Interpersonal conflict at work: an analysis of behavioral responses. *Human Relations*, **42**, 757–770.

Wall, D. (1987) Small group conflict: a look at equity, satisfaction, and styles of conflict management. *Small Group Behavior*, **18**, 188–211.

Walz, D., Elam. J. & Curtis, B. (1993) The dual role of conflict in group software requirements and design activities. *Communications of the ACM*, **36**, 63–76.

Zachary, G. (1994). *Showstopper: the Breakneck Race to Create Windows-NT and the Next Generation at Microsoft*. The Free Press, New York.

Zachary, G. (1998) Armed truce: software in the age of teams. *Information Technology & People*, **11**, 64–69.

Zultner, R. (1993) TQM for technical teams. *Communications of the ACM*, **36**, 79–91.

Bibliography

Steve Sawyer is an associate professor on the faculty of the School of Information Sciences at the Pennsylvania State University. His research interests encompass social and organizational informatics and, in particular, the social processes of software development, systems implementation and related organizational changes. Steve earned his doctorate at Boston University and has also served on the faculty of Syracuse University's School of Information Studies. He has published papers, or has them forthcoming, in journals such as *Communications of the ACM*, *Computer Personnel, IBM Systems Journal, International Journal of Information Management* and *Information Technology & People*. With co-authors, Rob Kling, Holly Crawford, Howard Rosenbaum and Suzie Weisband, his first book, *Information Technologies in Human Contexts: Learning form Social and Organizational Informatics*, is due out in 2001.

Appendix A: Indicators used in this study

This appendix presents the indicators used in the study. They are organized by factor and the source is listed in the brackets. All indicators are drawn from previously developed scales. The indicators were randomly ordered in the survey. The developer survey contained the indicators for characteristics of the project, the team, team conflict and conflict management. A second survey, for stakeholders, contained the software development team indicators. Both surveys used a seven point scale where one represents low/poor/worst whereas seven represents high/excellent/best. Reverse coded means that the response to this indicator was reversed to match the directionality of the others.

CP: characteristics of the project (Guinan *et al*., 1997)
Our development team gets all the information we need to plan our work (reverse coded).
When our development team encounters a difficult operational problem, it is hard to get the technical assistance we need.
To what extent do requirements/design change requests occur in your work?
To what extent do multiple views exist of how the final system should be developed?

CT: characteristics of the team (Guinan *et al*., 1997)
Team members have developed effective plans and procedures to co-ordinate work.
This team establishes clear expectations about how team members should act.
Everyone in our teams cares about the group.
Members of this team clearly view themselves as a team of people who work closely together.
Team members care a great deal about this team.

CTM: characteristics of the team member (Guinan *et al*., 1997)
I believe this team will really inspire the very best in me in the way of job performance.
Team members are highly dedicated to this project.
Team members have a comprehensive understanding of the user's business processes.
Team members know a lot about the business function area in which this system will be used.

TC: team conflict (Green & Taber, 1980)
During the systems development process, do you find that there is more than one satisfactory solution for problems faced?
The people in this team get on my nerves (reverse coded).
There is a lot of unpleasantness among the people in our team (reverse coded).
Dealing with the members of this team often leaves me feeling irritated and frustrated (reverse coded).

CM: conflict management (Green & Taber, 1980)
This team resolves the differences that exist among team members in a timely fashion.
This team finds ways to minimize tension between team members.
Team members do a good job of co-ordinating their activities.

TP: stakeholder-rated software development team performance (Henderson & Lee, 1992)
Efficiency of project team operations.
Quality of the system produced by the project team.
Adherence to schedules during the project.
Amount of work the project team produced.
Ability of the project team to meet the goals/commitments of the project.
Extent to which the system adds value to our firm.
The extent to which the system adheres to organizational standards.
Extent to which the users' business needs are reflected in the system.
Number of defects in the system.
The contribution of the system to the performance of the firm.

Appendix B: tolerance

Tolerance refers to the portion any given variable does not share with all other independent variables. Tolerance values range from 0.0 to 1.0 and 1.0 means total dependence (Pedhazur & Schmelkin, 1991, p. 436).

Full model	
Characteristics of the project	0.648
Characteristics of the team	0.544
Characteristics of the team members	0.709
Existing level of conflict	0.457
Low levels of conflict model	
Characteristics of the project	0.472
Characteristics of the team	0.552
Characteristics of the team members	0.554
Existing level of conflict	0.467
High levels of conflict model	
Characteristics of the project	0.583
Characteristics of the team	0.552
Characteristics of the team members	0.754
Existing level of conflict	0.669