v 4.1 as of 5/26/98

Software Development: Processes and Performance

A slightly modified version of this paper appeared as: Sawyer, S. and Guinan, P., (1998) "Software Development: Processes and Performance," *IBM Systems Journal*, *37*(4), 552-569.

Steve Sawyer
Syracuse University School of Information Studies,
4-102 Center for Science and Technology,
Syracuse NY 13244-4100.
Fax: 315-443-5806
(o): 315-443-4473
(e): ssawyer@cat.syr.edu

P.J. Guinan
Babson College
Babson Park MA 02157-0310
Fax: 617-239-6416
(o): 617-239-6462
(e): guinan@hc001.babson.edu

Acknowledgments

The assistance of the Heartland developers, Bob Spillers, Joel Farber, Harry Campbell, Dave Tolleson, Mike Pauser, and Mike Dockter have made this paper possible. Comments on earlier drafts from Bob Benjamin, Lisa Covi, Kevin Crowston, Bob Heckman, Chatpong Tangmanee, Ping Zhang and three anonymous reviewers have substantially improved this paper.

Biographies:

Steve Sawyer is an assistant professor at the Syracuse University school of information studies. Dr. Sawyer received his doctorate at Boston University. His research focuses on how people work together and how they use technology. Present research includes investigating how software development can be improved through attending to the social aspects of working together and the differences between packaged and custom software development. He has published in *Computer Personnel*, the *IBM Systems Journal*, *IEEE Software*, and *Information Technology & People*.

Patricia J. Guinan is an associate professor of information systems in the mathematics and science department at Babson College. She is the MCDermott Term Chair recipient at Babson and conducts both applied and theoretical research in the areas of technology transfer and communication-related issues in information system design. She received her Ph.D. from Indiana University. Her research has been published in a number of journals, including *Human Communication Research*, *IBM Systems Journal*, *Group and Organizations* and *Information Systems Research*. Dr. Guinan has also published an award winning book entitled *Patterns of Excellence for IS*

Professionals.

Software Development: Processes and Performance

Abstract

This paper presents data describing effects on software development performance due to both the *production methods* of software development and the *social processes* of how software developers work together. Data from 40 software development teams at one site that make commercial (or shrink-wrapped) software are used to assess the effects of production methods and social processes on both software product quality and team performance. Findings indicate that production methods, such as the use of software methodologies and automated development tools, provide no explanatory power about variations in either software product quality or team performance. Social processes, such as the level of informal coordination and communication, the ability to resolve intra-group conflicts, and the degree of supportiveness among the team members, can account for 25% of the variations in software product quality. These findings suggest two paradoxes for practice: (1) that teams of software developers are brought together to create variability and production methods are used to reduce variability and (2) that team-level social processes may be a better predictor of software development team performance than are production methods. These findings also suggest that other factors, perhaps other social actions and/or individual-level differences, must account for the large and unexplained variations in team performance.

Software Development: Processes and Performance

This paper presents data describing effects on software development performance due to both the *production methods* of software development and the *social processes* of how software developers work together. The premise behind this paper is that the production methods of software development such as methodologies, techniques, and tools come and go (at an ever-increasing rate) while the social processes of the people developing software change more slowly. For example, there has been a stream of software development methodologies (e.g., object-oriented, clean-room, rapid-prototyping), techniques (i.e., participatory design, requirements engineering) and software development tools (such as computer-aided software engineering, or CASE, tools) which reflect a production focus on software development. At the same time our knowledge of how software developers work together is growing more slowly [e.g., 89, 10, 93].

Since software development is, at the least, partly a social process means that understanding how people work together to build software is critical since software's importance in our society is matched by the difficulty encountered in its development. For example, about 40% of US corporate capital expenditures are directed towards software [70]. The Federal Government is also committing billions to supporting research in computer hardware and software [39]. Large- scale failures of software underscore the difficulty in its development [34]. For example, the IRS has spent billions to replace the present tax system with little to show and no replacement [80]. Further, corporate-level failures of information systems are routinely reported in the popular press [e.g., 31, 32, 86].

In this paper we draw on data collected from 40 software development teams at one US site [1]of a large, global, software and hardware manufacturing company (Compuco). These data are used to

[1] The research design decision is discussed in a subsequent section.

explore the relationships of both the production methods and social processes to software development performance. Specifically, in this paper we address two questions:

1. *What are the contributions to software development performance due to production methods?*
2. *What are the contributions to software development performance due to the social processes of software developers working together?*

To address these two questions the paper continues in four parts. The first part highlights issues with developing software and how we address these. The second part highlights our data collection. The third part presents our analysis and findings. The paper concludes with implications and suggestions for software developers and for software development researchers.

A MULTI-PERSPECTIVE VIEW ON DEVELOPING SOFTWARE

This part outlines issues with software development in general and the particular issues at the research site used in this study. This discussion is premised on the observation that the two most common responses to the current difficulties with making software are for software development organizations to: (1) establish and follow more formalized production methods for building software products and (2) use teams of software development specialists and the potential positive synergy which arises from their interactions. Typically, software development teams are brought together to make a new-- or enhance existing -- products. A team means two or more software developers who are engaged in building a defined product to be delivered within a certain time-frame. A team relies on the collective skills of its members because the scope of the effort, the inherent complexity of the effort, and number of tasks needed to develop modern software normally exceeds the ability of any one developer.

These team's members rely on methodologies, techniques, and tools to support software production. A methodology represents the set of tasks, and their ordering, that sets out the processes of production. For example, rapid application development (RAD) [53, 12], clean-room [61], object-oriented

[8] and formal [30] methodologies are currently popular. Techniques are sets of actions taken to complete a particular task. Examples include Joint Application Development (JAD) [90]and structured walkthroughs [29] Typically, a methodology draws on many techniques. A tool provides automation or structure to a technique [82] This is the premise behind CASE tools: they are not a methodology by themselves, they support the use of techniques and can enable methodologies [37 , 46].

This *production* perspective highlights the present emphasis for much of the current research on software development. Production methods describe how individuals will work and the focus is on the project, with the team's goal being to follow the proper method and use the proper techniques and tools in support of that project [20]. A substantial body of writing focuses on the role of methodologies, techniques, and tools (e.g, 66, 57]. The production perspective seeks to underscore the repetitive, predictable, and manufacturing-like aspects of software development [e.g., 44, 45, 64]. Reducing the variations in, and routinizing, the development process, developing specific guidelines as to how to best address common tasks, and making tools available to support these efforts are seen as central to the maturation of software development as a profession and as a more certain contributor to society [23, 88, 4].

A second perspective on software development is as a *social* process [62, 43, 73, 75]. This perspective's focus is on how software developers work together to produce software. The trend to using teams to produce software magnifies the issues of working together to build software [19]. For example, the social issues of working together to build software include coordination and communication breakdowns [19, 9, 10, 89] and the positive and negative effects of such social processes as intra-group conflict management [71, 72, 87]. Recently, the culture of United States (US) software development has been seen as unifying force among developers, providing them a commonality of focus that enables software development [14, 92]. This implies that another issue for software development teams is the degree of shared norms that tie software developers together.

Social processes of software development encompass both informal communication and intra-group coordination activities such as discussing how activities will be performed, finding -- and taking -- the time to talk with other team-members, and the sharing of ideas and information between each other [50, 81]. Social processes also include resolving the conflicts that arise in the course of working together and encompasses issues such as reducing the level of irritation and frustration among the team, getting along well with each other, and resolving differences in a timely manner [83, 69]. There are also norms of loyalty and supportiveness that are part of the enculturation of software developers as they learn how to work together while producing. This includes caring for other members in the group, the occurrence of helping behaviors between team-members, being inspired by being a member, and seeing the team as distinct from the organization.

A third way to view software development is from an *individual* perspective. This perspective focuses on the unique contributions of individuals to the team. Issues from an individual perspective can include, for example, the degree of experience and skill that each developer contributes [87]. The individual perspective also encompasses the internal motivations for participating (such as ego gratification, social power accumulation) and other personal motives [89].

In this multi-perspective approach to understanding software development, distinguishing between the production, social, and individual levels of a software development team's work is analytically useful, but these three perspective's actions occur in a tightly woven, interdependent process. [59, 60]. For example, including formal coordination mechanisms as a part of the production process, while informal communication and coordination mechanisms are considered part of the social processes, is an analytic distinction. In both formal and informal meetings, team-members socialize. They talk, they encourage (or discourage) supportive feelings towards the team, they respond (or ignore) potential intra-group conflicts, and they do (or do not) communicate and coordinate. Further, the team-members jockey for social power, they worry about the effects of embarrassing questions, and they try to impress their peers. Thus, the formal coordination factor reflects an intersection of production, social, and individual processes. However, for this study, we focus on the team-level issues of production and social perspectives, leaving the individual perspective for future work. We return to this issue in our discussion of the findings at the end of the paper.

A fourth perspective on software development is *contextual*. This perspective suggests that issues such as the competitiveness of the company, the industry in which the company operates, the degree of managerial skill, the level of resources, and other extra-organizational factors affect software development performance [19, 15]. For example, these factors contribute to the difficulty in comparing data about software development teams who focus on making custom-designed embedded software for the United States' Department of Defense (US DOD) with teams building DOS/Windows-based packaged products for the personal computer market. That is, the differences in customers, markets, and goals of the software product are so diverse that comparisons are often difficult [15]. Because of the potential for contextual issues to overshadow the goals of this study, we focused on collecting data from one organization. This provides some control (albeit imperfect) of the potential effects of organizational and environmental context. It also reduces (but does not eliminate) the potential effects of intra-departmental differences inside the.

The Setting: Developing Software at Heartland

The data we report on in this paper are drawn from a field study at one US software development site (Heartland) of Compuco. Heartland makes sub-system software such as database products and languages. These are sold as packages, often in combinations that can provide for integrated solutions, in the commercial market. Packaged software is also known as commercial, shrink-wrapped, and commercial-off-the-shelf (COTS). As a packaged software vendor Compuco licenses their product for use by others. These products may have thousands, or even millions, of licensees. This implies that development of these products is done for a distant user who is typically not a member of the organization, making extensive user/developer interaction difficult [48]. Further, the norm at Heartland is for developers to be involved with one product for some time, participating in the product's developmental trajectory [13, 92]. This means that the developers become quite knowledgeable about both the product's features and the development history. Finally, and unlike most custom IS development efforts, Heartland's developers have no hand in their product's implementation [15].

Packaged software development, here exemplified by Heartland, is an expanding and important aspect of the software industry. While the US DOD, and other large institutions, may always demand a certain level of custom-built software, the market for packaged software is both large and growing [14, 70]. For example, IBM has been a dominating force in commercial software with products such as IMS and DB2. Microsoft's rise as a corporate power is built on the sales of its packaged software [92, 22].

Three reasons made Heartland a suitable research site. First, from above, they develop products for the commercial market. As noted, this is an important, expanding, and relatively under-studied area of software development. The second reason is that they are large enough to support enough teams to provide adequate data for the study. Finally, the software developers and managers were motivated to participate. For example, like other commercial software developers, Heartland's products face increasing competition. Product time-to-market issues and product quality are often competing pressures. The market life of each commercial software product being developed is also shortening. Even so, long-term maintenance demands increase with each new release. This combination led to difficulties in maintaining existing products and slowed the pace of new product development at the host research site. Concurrently, developers were increasingly unhappy with the way their work was structured and with the work-life pressures they faced. They were working harder, spending more time at work, and seeing fewer results. Senior managers at Heartland were also under tremendous pressure from the Compuco corporate executives to create new products and extend revenue streams on existing lines.

To respond to the intertwined influences of market pressure, product pressure, work-life pressure, and corporate pressure, Heartland's senior development managers responded by re-focusing development to be team-based. This was also a move away from the functional/ project matrix management structure that they had relied on for more than 20 years. As a part of that effort, this research represents a joint effort between the authors and the developers at Heartland to better understand the software development team processes and performance effects at their site. However, management and team structures remained unchanged during the course of this study. The current team structure is based on having a team leader, a technical leader, and a relatively small number of developers to comprise the entire team. Each team's leader reports, in turn, to a product manager.

The Production Aspects of Software Development

Heartland's software development organization exemplifies the *production* perspective on software development. The site follows a rigorous, well-defined software development method which is based on structured analysis and design approaches and has been evolving for nearly 20 years. This methodology is both well-known, and heartily supported, by senior development managers. There is extensive training in the core software development methodology and base techniques provided to all developers. As part of this method's development, personnel have developed and integrated a number of automated software development tools (some made in house, some commercial products themselves) into the development methodology. This methodologic rigor is, in part, one of the reasons why Heartland has been lauded for their process quality (having won both several internal-to-the-company and industry-wide quality awards in the past few years). However, even at a single site, where developers have been trained in one set of techniques and share a common set of tools, the way these aspects of production are used varies.

To capture the use of, and variations in, the production aspects of software development at Heartland, we collected data on: the level of formal coordination mechanisms use, the level of formal methods use, and the levels of tool use. Formal coordination mechanisms includes things such as formal project meetings, formal client meetings, and required documentation/deliverables. Thus, this measure of formal coordination acts as a surrogate of project management. The other two factors represent the use of a software development methodology. For instance, the use of version control procedures and management of code libraries suggests a reliance on a defined software development methodology. High levels of tool usage reflect the use of standard development techniques which allows for automation.

The Social Processes of Software Development

The second perspective on software development is as a *social* process. This perspective's focus is on how the software developers work together to produce software. Heartland’s development organization also recognizes the importance of the *social* processes of software development. For example, there are numerous group dynamic, conflict management, and listening-skills seminars, extensive rewards for superior team performance, and both formal and informal acknowledgments that teamwork is critical. Further, development team members meet on a regular basis (typically weekly) and are normally physically co-located.

The social processes of software development that we measured include how team-members perceive their level of informal communication and coordination, their ability to resolve intra-group conflicts, and the degree of supportiveness and loyalty they felt for the other members of their team. The level of informal communication and coordination means the amount, and value of, communication both with team-members and with key people who are not part of the team. The ability to resolve intra-group conflicts includes both surfacing differences and negotiating acceptable shared agreements and/or compromises. The degree of supportiveness includes how team-members feel toward other members of their team, and how they perceive the team-members feel towards them.

Software Development Performance

To assess the value of production and social processes requires a measure of software development performance. This is problematic for at least two reasons. Firstly, the performance of software development teams has been measured in many different ways. Secondly, most of these measures have significant measurement problems such as limited relationships to business value and questionable validity [49, 24].

Our approach is to view software development as an activity intended to produce a product that will affect the behavior of one or more stakeholders. We further constrain our definition of stakeholders to be individuals who are *not* team members but who can affect design activities and who can be affected by the resulting information systems [77, 54, 41]. While the development team members certainly have a stake in the product, our focus is on the external-to-the-team stakeholders. These might be user managers, senior development managers, and/or senior customers and they assess the team's performance based on their knowledge of the organization's needs, experience with previous and ongoing software development projects, and their expectations for quality. Thus, we share the view of Seidler [77] who finds that perceptual assessments of performance provided by such knowledgeable managers (i.e., stakeholders) have a high level of convergence with other objective measures of performance.

Given these issues, we characterize software development performance as multi-dimensional and include three attributes: product quality, team efficiency and team effectiveness. Stakeholders provide the product quality assessment. Both the team effectiveness and efficiency measures are also assessed by stakeholders and combined into an aggregated measure of team performance. The team performance factor is also drawn from the developers on each of the teams. This provides a self-reported measure of team performance in terms of team effectiveness and team efficiency. Thus, three performance factors are measured: stakeholder-rated product quality, stakeholder-rated team performance, and self-reported (by the developers) team performance.

STUDYING HEARTLAND'S SOFTWARE DEVELOPMENT TEAMS

This part describes the research methods used to assess the effects of the production and social processes of software development teams on software development performance. To do this, we used multiple data collection methods: direct observation of their work as it occurred, quantifiable data on use and performance (drawn from two surveys), and a synthesis of the anecdotes and stories of software development. Beginning in early 1993, we spent 18 months collecting data on Heartland's software developers doing their work, in their native environment. Using interviews and surveys is sensible since we are collecting data on their perceptions about how their teams rely on the various production process and social process factors. The individual responses to the survey are aggregated to the team level for analysis [47]. Analysis of the survey data is done using multiple regression/ correlation techniques. [1, 26, 16, 65]. This means that each performance factor is assessed for the contribution of both the production and social process factors (represented as the amount of variance explained). Table 1 presents these factors and their definitions. Appendix A presents their zero-item correlations, means, and standard deviations.

With the support of the team members and their managers (including senior managers), we gathered data from 45 software development teams. We formally interviewed 56 people. Many of these people spoke with us again, often several times, over our observation period. We also met informally, or in organized response sessions (such as debriefings of teams), with another 153 developers and managers from the 45 teams following our survey data collection. The topics of these informal meetings varied, but the issues and questions were driven by the ongoing analysis of both survey data and previous interviews. Most of the informal sessions were not taped. Instead, these were documented with post-hoc field notes [7].

Survey-based data were gathered from 40 teams (and 128 respondents) at the site. This represents more than 30% of the project teams and 10% of the developers at the site. Five of the 45 teams were not surveyed: three declined and two were performing software support and not development. The surveys were developed from existing scales (see Note 1) and pilot-tested at the site [25]. We used these surveys to gather data about how the team members interacted with each other, how they produced software, self-reported performance, and demographic data about the teams and team-members. Survey questions were posed at the team level as this is the level of analysis. This is also the level of measurement and the level of theory [51].

We also collected data about performance from stakeholders for each of these teams. This was done with a structured survey in a phone-based interview employing the scale developed and used by [41] in their study of software developers. In the debriefing (offered to all teams and accepted by 28) which followed the major data collection and analysis phases, we returned to the site and asked additional questions to reflect on and amplify the findings from the initial data analysis.

Some additional data on product quality were gathered by the site and provided to us. This archival data on product quality is at the product level. This cannot be directly linked to the individual modules. This means that product quality data cannot be associated with the 40 teams in this analysis. However, product quality data can be compared at the product level and that is discussed in the final part of this paper.

Interview data are used to amplify findings from the statistical analysis. Self-reported team performance data is drawn from the same survey which is used to collect the predictors of that performance. This is a form of method bias that typically results in over-exaggerated relationships [65]. Thus, the regressions based on self-reported team performance may be more useful as an indicator of a relationship's existence than as a measure of that relationship's strength. Stakeholder data are used to develop the team performance and product quality factors. These data are drawn from separate surveys and are not as susceptible to this method bias.

The Heartland Developers

As we noted at the beginning of this paper, the development teams in this study build software for commercial sale. These commercial software products are quite large (typically more than one million lines of code) and some have been in the market for three decades. The teams involved in this study contribute to one of four products. Each team is typically charged with one module – a distinct part of the overall product – that must be integrated together and work seamlessly in the final assembled product. What this means is that, while the modules may be distinct segments and identifiable at some level, as a product they are tightly integrated into one system. During development, this means that teams are often highly dependent on one-another and these dependencies are not sequential. That is, two modules may pass data back and forth when used in the product. This means the two teams must work closely together as both customer and producer. Team size ranges from four to 14 people, with the average being nine members. On average, the team members have been together for nearly two years and they change team leadership every year. Table 2 includes more information about these developers.

The level of experience and product knowledge are viewed as critical measures of a team's competence. For example, developers on the 40 teams have a mean of more than nine years in software development, more than four years with the company, and nearly two years with their present teams. The individual respondents have a mean of 9.8 project's worth of experience (where a project means a previous release of a product). Eighty-nine percent of the respondents have a college degree; 34% also have a masters or doctorate. Many of the original developers have stayed with these products since inception and they provide a wealth of product knowledge and perspective to the teams. Further, an espoused belief at the site is that it takes several years for junior developers to become fully aware of product intricacies.

EFFECTS OF PRODUCTION AND SOCIAL PROCESSES IN SOFTWARE DEVELOPMENT

To address the two questions posed at the beginning of the paper, this part presents the analysis and findings in four sections. The first section describes the effects of production factors on software development performance. The second section presents the added effects of the social process factors on software development performance. The third section describes the moderating influence of production factors such as methodology and tool use. The fourth section presents issues with the analysis.

The Effects of Production Processes on Software Development Performance

Tables 3 and 4 present analyses that assess the contributions to software development performance due to production methods. Table 3 presents the contributions of the two methodology factors to each of the three performance factors. This analysis shows that both the formal method and tool use factors provide no significant contribution to any of the three performance factors. That is, variations in the use of formal methods and/ or the use of automated development tools provide no explanation of the variance in stakeholder-rated product quality, stakeholder-rated team performance or self-reported team-performance.

Table 4 presents the contribution of the two methodology factors and the formal coordination factor to each of the three performance factors. The formal coordination factor accounts for a significant portion of the variance in both the stakeholder-reported and self-reported team performance. However, none of these production factors provide any significant explanation of the variation in stakeholder-rated product quality.

This analysis indicates at least two findings. Firstly, the methodology factors provide little direct explanatory value. Secondly, the use of formal coordination mechanisms helps to explain the most variance in two of the three performance factors. What this also suggests is that these production factors may have an indirect, or moderating, effect on performance [84, 16]. This means that the use of formal methods and/or automated software development tools may influence the relationship between the social process factors and the performance factors. This will be tested, below.

The Effects of Production and Social Processes on Software Development Performance

To assess the contributions to software development performance due to the social processes of software developers working together, Table 5 presents results of the analysis combining both the production and social process factors in assessing their contribution to each of the three performance factors. Data show that higher levels of formal coordination, higher levels of informal communication, more intra-group conflict management, and higher levels of supportiveness explain 69% of the variance in self-reported team performance. The data in Table 5 also show that higher levels of formal coordination, higher levels of informal communication, and more conflict management account for 23% of the variance in stakeholder-rated team performance. These factors also account for nearly 20% of the variance in the quality of the team's software product.

The Moderating Influence of Methodology and Tool Use

Table 6 presents additional analysis into the role of production factors in explaining variations in the three performance factors used. This was suggested by the findings reported in Table 4. To conduct this analysis, we split the sample of teams into two sub-sets. The first sub-set rated lower than the aggregated mean of the use of formal methods and automated software development tools. The other sub-set rated higher than the aggregated mean of both factors. Regressions for each of the three performance factors using the remaining four factors for both sub-sets produced no significant models of the teams with high formal method/ tool use.

Two significant models were found for teams in the *low* formal method/tool use sub-set. That is, for the 20 teams in the sub-sample that had lower levels of methodology use, the levels of formal and informal coordination and conflict management are significant contributors, explaining 25% of the variance in product quality. All four factors are significant contributors and account for 62% of the variance in self-reported team performance. This implies that, for the teams who make minimal use of both formal methods and/or automated software development tools, low-levels of use moderate the relationship between the three social process factors (plus formal coordination) and two of the three performance factors used in this analysis. Further, the performance of teams who used both formal methods and/ or automated software development tools the least are not significantly different than the high use teams.

Similar split-sample analysis using the social process factors and formal coordination produced no significant models. This suggests that these social process factors and formal coordination do not have any moderating influence on the use of formal methods and automated software development tools. Since two of the performance factors are based on the perceptions of stakeholders, it is important to note that these stakeholders were unlikely to know much about the social processes of the teams. However, they would be aware of any formal coordination mechanisms which required the team -- or its leaders -- to contact the stakeholder (via either meetings or deliverables).

Analysis Issues

There are at least two issues with this analysis that merit additional discussion. The first issue regards the use of multiple regression as the basic analysis technique. That is, analysis based on multiple regression is susceptible to multi-collinearity among the independent variables [1]. Multi-collinearity is the tendency for supposedly independent factors to be related to each other, possibly invalidating certain assumptions behind using this statistical technique. Potential multi-collinearity issues are minimized if the 'tolerance' between any two factors remains below 0.700 [65]. In this analysis no tolerance exceeded this upper limit.

A second issue with this analysis regards the sources of data. Since all data are collected at one software development site, there may be some concern with the representative distribution of the data. Appendix A provides the means and standard deviations for all the factors used in this study. The normal distributions (indicated by the standard deviations) suggests that the use of one site for data collection did not constrain the range of responses used to construct the factors used in this analysis.

THE ROLES OF PRODUCTION FACTORS IN SUPPORTING THE SOCIAL PROCESSES OF SOFTWARE DEVELOPMENT

Data indicate that the level of both formal methods and automated tool use do not aid in predicting software development performance. However, the role of formal coordination, and several of the social process factors, can account for some of the variation in the software development performance we measured. For example, the combination of production and social process factors accounts for 25% of the variance between the teams with the highest and lowest levels of stakeholder-rated product quality.

In this fourth part we address four issues suggested by this analysis: (1) rethinking the relationship between the production and social factors and (2) potential limitations due to the interesting sample demographics. The final two sections go beyond the data from the current study to present discussions of: (3) the potential effects to software development practice suggested by these findings, and (4) the question of the unexplained variance in stakeholder-rated team performance and product quality.

Rethinking the Relationship between Production and Social Process Factors

Data from this study show that of the three production factors measured, the two which focus on assessing the methodologic aspects -- level of method and tool use -- are not significant predictors of software development performance. Furthermore, the analysis of the moderating effects of both methods and tool use in this analysis suggest that higher levels of method and tool use are even less valuable than lower levels in helping to predict software development performance.

One reason this site was selected for the research is that they have a well-established software development methodology. This is accompanied by extensive training and the provision of automated tools to support steps of the software development methodology. Data presented in Table 7 indicate that many of the method aspects are not used extensively and are seen as having marginal value. Further, while tool use varies, it provides no predictive value for performance. Other data, provided by the site on quality at the product level, provides little additional information. For example, the distribution of the teams, relative to all three measures of software team performance, does not differ across the five products to which these 40 teams contribute.

The findings suggest that one explanation for the importance of the formal coordination factor is it reflects actions that bring the production aspects of software development into a social context. That is, this factor represents the use of meetings, documents, and required interactions that bring the developers together. In this way, the formal coordination aspects of a methodology are valuable since they provide for *an occasion to socialize*. Observations of meetings and the anecdotes of participants suggest that it is the process of socializing at these meeting that provide value, not the occurrence of the meeting or the requirement of the methodology being followed to have that meeting occur. This finding also implies that a value of project management, for which this formal coordination factor serves as a surrogate, may be that most project management techniques require team-members to interact .

If team-level software development methodologies serve primarily as a means to require socialization, it may be that these methods are important primarily because they help to teach team-members when, and about what, to discuss. In that sense, teaching software development methods and tool usage may have the unintended effect of guiding 'valued' social processes. For example, Vessey, and Sravanapudi [85] find that CASE tools serve as effective collaboration devices. This is one way to interpret the value of lower-levels of method and tool use: they reflect an effective source of guidance without dogmatic (and socially counter-productive) effects that may arise from over-adherence to formal methods or tool use.

The explanatory value of both social processes and the role of formal coordination highlight the importance of non-production aspects of the development process. While this has been widely recognized, this analysis underscores the extent to which the factors influencing software development performance are still poorly understood [23, 34]. Further, this finding suggests that *allowing* for "people factors" in the discussion of new software methodologies [e.g, 55, 13] may be placing the emphasis on the wrong aspect : putting the cart before the horse. Perhaps software development methods should be developed to explicitly encourage socialization among developers -- a behavior-centered process? This is the gist of Tom DeMarco's [27] point that most software developers, when asked to work with another on a project, never ask, "in what language?," they ask, "with whom?" Refocusing software development methods from production (or engineering) -centered to social (or behavior) -centered seems appropriate given that the data show that software development production aspects are both secondary to social aspects of software development and most valuable if used sparingly [18].

This emphasis reversal, where production methods are designed to support the social processes of software development teams, is well-represented in the discussions of software development at Microsoft [21, 22, 92, 93]. They contend that Microsoft's development processes are based on individual interaction and flexible processes based on fixed team meetings. Work by Zachary [92, 93] suggests that the social interactions of dominant team-members shape the development processes more than do production methods. Sawyer, Farber and Spillers [75] find that, when allowed, software teams will adapt tool use to fit their own, emergent, needs. Curtis [18] asserts that any software development effort that does not explicitly account for how people work together is likely to be unsuccessful. Data from this study support these assertions and findings.

In discussing the limited effect of methods and tool use on software development team performance, it is imperative to realize that our focus is at the team-level uses of tools and methods. Thus, the steady evolution and improvement of programming languages, compilers, debuggers, and other elements of individual-level software development do not contradict these findings. In fact the improvements in software development at the individual level have been substantial. The issue raised by this analysis is that the improvements in individual productivity due to individual-level tools, which *do* help software developers, are only *indirectly* linked to team performance. The same is not as certain for methods that focus on organizing the group [i.e., 42]. This data suggest that this absence of affect arises because current methods are not developed around the social interactions of developers, focusing instead on producing software. Since the three levels of software development -- individual, social, and production -- are tightly tied together, a myopic focus on production actually decreases the potential value which software development methodologies are to provide.

Interesting Sample Demographics

One issue with generalizing the findings from this study is the potential uniqueness of the sample. Firstly, these developers make packaged software and this is different from traditional information systems development [15]. Secondly, data in Table 1 suggest that three characteristics of the developers at this site are unusual: level of formal education, years of professional experience, and team stability (as measured by time in the same job and time as a member of the same team). This combination of extensive formal education, professional software development experience and team stability suggests that, while the production and social aspects of software development provide some predictive value, the major differentiating factors may be at the individual level.

These data imply that, even with well-developed communication, coordination, conflict-management skills and a strong sense of team supportiveness, the performance of software development teams hinges on individual talent. This scenario lends additional credence to Boehm’s [5] work with software cost drivers. He suggested that individual programmer differences are three times the power of the team’s effect on software costs. Weinberg [89] posits that the difference between the best and worst programmers may be a factor of 10. One commonly held belief is that software gurus and tremendous individual contribution are the basis for commercial software success [e.g., 92, 93]. The current study provides data to support this assertion. What is not clear is the extent to which the social aspects of software development mitigate or enhance individual skills.

Paradoxes for Practice

Generalizing to a broader population based on the data from this small, and possibly unique, sample must be seen as speculation. That said, we speculate that the findings of this study suggest two paradoxes. The first paradox is the need for teams of software developers to provide diverse perspectives versus the use of software development methods that are designed to minimize variance. That is, one of the premises in establishing software methods and formal production processes is to remove some of the variance that working together in the highly dynamic, conflict-oriented, pressure-filled, work setting which characterizes commercial software development seems to demand.

In contrast to the cross-functional, heterogeneous basis of teams, formal software processes are designed to remove individual variability. This is done by focusing on *how* deliverables occur, implicitly ignoring *who* make them. The premise behind using teams in software development is that the production losses due to the need for people to work together are, it is believed, offset by the production gains of this group effort [79]. This means that a defined software development process is designed to reduce the individual variability that the use of teams comprising multiple software development specialists is supposed to enhance. The data in this study indicate that, while a defined software development method may not directly contribute to software development performance, its absence heightens the effects of the social processes.

A second paradox is that the skills surrounding the social process issues of coordination, communication, conflict management and supportiveness are typically not being provided to developers as part of their formal education. So, these skills are either developed on the job or the developer is limited by their ability to bring their technical knowledge to bear in the social world that is software development. Given the increasing technical demands, and the quick-changing nature of computing technology, the commercial software developer is often caught between maintaining and expanding their technical skill-base while also being required to learn and use these 'soft-skills.' Since software developers have relatively low levels of social needs [17, 94], this increased emphasis on socialization may be producing part of the stress which Heartland's developers are experiencing.

This skills paradox suggests that embedding mechanisms to improve the social processes between developers can aid in developing both sets [75]. Curtis [18] asserts that there is still too little attention paid to developing methods of software development based on the socialization patterns and needs of the developers (though Microsoft's synch-and-stabilize approach does so implicitly [21, 22]). Certainly practicing developers understand the need for interaction and communication at an informal level [89, 34]. As Pressman [67, p. 18] observes about the obvious, "A process that stifles creativity will encounter resistance."

A third issue is suggested by the data in Appendix A. The correlations among the three social process factors and formal coordination suggest that these four factors may represent facets of a higher-level factor. That is, these four factors may be representing a higher-level factor such as “Software Development Team Culture.” This nebulous concept is both acknowledged in practice and not easily measured [14,15]. In that context, this research represents a small step toward a better understanding of the norms, behaviors, myths, and rituals that make up the culture of software development, a culture which extends beyond the realm of production [3, 14].

What About the Unexplained Variance?

We set out to explain the performance variations in software development due to both production and social factors. Analysis of the survey data from the 40 software teams at Heartland suggest that other, unmeasured, factors must account for the unexplained variance. We focused on the social and production levels of software development teams. Findings from this study suggest that we should now include individual-level factors such as motivation, experience, and knowledge [e.g., 34, 87, 89]. At the social level, we could also expand the analysis to include other factors. For example, we did not account for team leadership and influence [91, 63], other aspects of group process such as boundary spanning [2], intra-group control [41, 50], social power [58], and more of the cultural aspects of development [76, 14].

From a production perspective, additional analysis might benefit from looking at the specific sub-cycles of software development tasks and focus in more detail on the use of specific techniques and tools. And, other measures of software development performance (such as lines of code or function points) may be more useful (i.e., [36] used a combination of reported and measured factors). Further, while there will always be random chance, we believe that it plays a smaller role than, for example, the 75% for stakeholder-rated product quality. The better accounting for variance using the self-reported team performance measure (versus stakeholder-rated) is, as stated, due in part to the use of one instrument to collect both process and outcome data. Finally, extending the analysis to allow comparisons across organizational boundaries may help to explore contextual factors that might influence software development. Given all these issues, what this research suggests is that we are still unsure of many of the key contributors to explaining the variance in software development performance.

REFERENCES

[1] Amick, D. and Wahlberg, H. (1975) **Introductory Multivariate Analysis**, Berkeley: MrCutchan Publishing Co.

[2] Ancona, D. and Caldwell, D. (1990) "Information Technology and Work Groups: The Case of New Product Teams, " in **Intellectual Teamwork: Social and Technological Foundations of Cooperative Work**, J. Gallagher, R. Kraut, and C. Egido (Eds), Hillsdale, NJ: Lawrence Erlbaum Associates.

[3] Avison, D. and Meyers, M. (1995) "Information Systems and Anthropology: An Anthropological Perspective on IT and Organizational Culture," *Information Technology & People, 8*(3), 43-56.

[4] Basili, V. and Musa, J. (1991) "The Future Engineering of Software: A Management Perspective," *Computer, 6*(5), pp. 90-96.

[5] Boehm, B. (1981) **Software Engineering Economics**, New York: Prentice-Hall.

[6] Boehm, B. (1987) "Improving Software Productivity," *Computer, 2*, 11, pp. 43-57.

[7] Bogdan, B. and Bicklin, S. (1982) **Qualitative Research for Education**, New York: Allyn and Bacon.

[8] Booch, G. and Rumbaugh, J. (1996) "Unified Method for Object-Oriented Development," Documentation Set 0.8, Rational Software Corporation.

[9] Brooks, F. (1987) 'No Silver Bullet: Essence and Accidents of Software Engineering,' *IEEE Computer, 20*(4), pp. 10-19.

[10] Brooks, F. (1974) **The mythical man-month: essays on software engineering**. Reading, MA: Addison-Wesley.

[11] Brown, S. (1995) "The Fall of Software's Aristocracy: Realizing the Potential of Development," in **The Future of Software**, D. Leebaert (Ed), Cambridge: MIT Press, pp. 157-175.

[12] Card, D. "The RAD Fad: Is Timing Really Everything?," *IEEE Computer, 12*(5), pp. 19-23.

[13] Carmel, E. and Becker, S. (1995) "A Process Model for Packaged Software Development, *IEEE Transactions on Engineering Management*,

[14] Carmel, E. (1997) "American Hegemony in Packages Software Trade and the 'Culture of Software'," *The Information Society, 13*(1), pp. 124-142.

[15] Carmel, E. and Sawyer, S. (1998), "Packaged software development teams: What makes them different?," *Information Technology & People, 11*(1), 7-19.

[16] Cohen, R. and Cohen, J. (1983) **Applied Multiple Regression / Correlation for the Behavioral Sciences**, Hillsdale, NJ: Lawrence Erlbaum Associates.

[17] Cougar, D. (1989) 'New Challenges in Motivating MIS Personnel,' *Journal of Information Systems Management*, pp. 36-41.

[18] Curtis, W. (1989) "Three Problems Overcome with Behavioral Models of the Software Development Process," Proceedings of the 11th International Conference on Software Engineering, 398-399.

[19] Curtis, B.; Krasner, H.; and Iscoe, N. "A field study of the software design process for large systems." *Communications of the ACM* 31, 11, (1988), 1268-1287.

[20] Curtis, W. Hefley, W. and Miller, S, (1995) "The People Capability Maturity Model: P-CMM," Software Engineering Institute Report (CMU/SEI-95-MM-01), Pittsburgh, PA.

[21] Cusamano, M. and Selby, R. (1997), "How Microsoft Builds Software," *Communications of the ACM, 40*(6), 53-61.

[22] Cusamano, M. and Selby, R. (1995), **Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People**, New York: Free Press/ Simon & Schuster.

[23] Davis, A. (1996) "It Feels Like Deja Vu All Over Again, *IEEE Software, 13,* 4, p. 4.

[24] DeLone, W. and McLean, E. (1992) "Information Systems Success: The Quest of the Dependent Variable," *Information Systems Research, 3*(1), pp. 60-95.

[25] Dillman, D. (1978) **Mail and Telephone Surveys: The Total Design Method**, New York: John Wiley & Sons.

[26] Dillon, W. and Goldstein, M. (1983) **Multivariate Analysis: Methods and Applications**, New York: John Wiley & Sons.

[27] DeMarco, T. (1995) **Why Does Software Cost So Much? And Other Puzzles of The Information Age**, New York: Dorsett House.

[28] DeMarco, T. and Lister, T. (1988) **Peopleware: Productive Teams and Projects,** New York: Dorsett House.

[29] Freedman, D. and Weinberg. G. (1982) **Handbook of Walkthroughs, Inspections and Technical Reviews**, Boston: Little, Brown.

[30] Fidge, C., Kearney, P. and Utting, M. (1997) "A Formal Method for Building Concurrent Real-Time Software," *IEEE Software, 14*(2), 99-106.

[31] Fisher, L. (1996a, August 8) "Data Network Suffers Biggest Blackout Ever," *The New York Times*, C5.

[32] Fisher, L. (1996b, August 9) "Human Error and Software Created Data Network Glitch," *The New York Times,* C1-3.

[33] Gause, D. and Weinberg, G. (1989) **Exploring Requirements: Quality Before Design**, New York: Dorsett House.

[34] Glass, R. (1997) "The Ups and Downs of Programmer Stress," *Communications of the ACM, 40*(4), pp. 17-19.

[35] Green, S. and Taber, T. (1980). The Effects of Three Social Decision Schemes on Decision Group Process. *Organizational Performance and Human Behavior, 25*, 97-106.

[36] Guinan, P., Cooprider, J. and Faraj, S. (In press) "Inside the Black Box of Software Development Teams," *Information Systems Research.*

[37] Guinan, P., Cooprider, J. and Sawyer, S. (1997) "The Effective use of Automated Application Development Tools," *IBM Systems Journal, 36*(1), pp. 124-139.

[38] Hackman, J. (1982). *A Set of Methods for the Research on Work Teams* (Technical Report 1). New Haven, CT: Yale School of Organization and Management.

[39] Hartmanis, J. and Herbert, L. (1992) **Computing the Future: A Broader Agenda for Computer Science and Engineering**, Washington DC: National Academy Press.

[40] Henderson, J. and Cooprider, J. (1990). Dimensions of I/S Planning and Design Aids: A Functional Model of CASE Technology. *Information Systems Research, 1*(3), 227-252.

[41] Henderson, J. and Lee, S. (1992) "Managing I/S Design Teams: A Control Theories Perspective," *Management Science 31*(6), 757-777.

[42] Hidding, G. (1997) "Reinventing Methodology: Who Reads it and Why?" *Communications of the ACM, 40*(11), 102-109.

[43] Hirschheim, R., Klein, H. and Newman, M. (1991). "Information Systems Development as Social Action: Theoretical Perspectives and Practice," *Omega, 19*, 587-608.

[44] Humphrey, W. (1995) **A Discipline for Software Engineering**, Addison-Wesley, Reading MA.

[45] Humphrey, W. (1988) **Managing the Software Process**, Addison-Wesley, Reading MA.

[46] Iivari, J. (1996) Why are CASE Tools Not Used?, *Communications of the ACM, 39*(10), pp. 94-103.

[47] James, L. (1982). “Aggregation Bias in Estimates of Perceptual Agreement,” *Journal of Applied Psychology, 67*, 2, 219-229.

[48] Keil, M. and Carmel, E. (1995) ‘Customer-Developer Links in Software Development,’ *Communications of the ACM, 38*(5), pp. 33-44.

[49] Kemerer, C, (1989) "An Agenda for Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts," Proceedings of the 22cnd Annual Hawaii International Conference on Systems Science,” pp. 219-228.

[50] Kirsch, L. (1996) “The Management of Complex Tasks in Organizations: Controlling the Systems Development Process,” *Information Systems Research, 7*(1), pp. 1-21.

[51] Klein, K., Dansereau, F. and Hall, R. (1994) "Levels Issues in Theory Development, Data Collection, and Analysis," *Academy of Management Review, 19*, 2, 195-229.

[52] Kraut, R., & Streeter, L. (1995). Coordination in Software Development. *Communications of the ACM*, *38*(3), 69-81.

[53] Lantz, K. (1989) **The Prototyping Methodology**, Englewood Cliffs, NJ: Prentice-Hall.

[54] Lee, S., Goldstein, D. and Guinan, P. (1991) "Informant Bias in I/S Design Team Research." in Nissen, E., Klein, H. and Hirschheim, R. (Eds). **Information Systems Research: Contemporary Approaches & Emergent Traditions**, Amsterdam: North-Holland 1991.

[55] Luqi and Goguen, J. (1997) “Formal Methods: Promises and Problems,” *IEEE Software, 14*(1), 73-85.

[57] Marciniak, R. (1994) **Software Engineering**, Thousand Oaks, CA: IEEE Press.

[58] Markus, M. (1983) 'Power, Politics, and MIS Implementation'. *Communications of the ACM, 26*(6), pp. 430-444.

[59] McGrath, J. (1990) "Time Matters in Groups, " in **Intellectual Teamwork, Social and Technological Foundations of Cooperative Work**, J. Gallagher and R. Kraut (Eds), Hillsdale, NJ: Lawrence Erlbaum Associates, 1-23.

[60] McGrath, J. and Hollingshead, A. (1994) **Groups Interacting With Technology**, San Francisco: Sage.

[61] Mills, H., Dyer, M. and Linger, R. (1984) “Cleanroom Software Engineering,” *IEEE Software, 2*(9), 19-24.

[62] Newman, M. and Robey, D. (1992) “A Social Process Model of User-Analyst Relationships,” *MIS Quarterly, 16*, 249-266.

[63] Nutt, P. (1986) 'Tactics of Implementation'. *Academy of Management Journal, 29*(2), 230-261.

[64] Paulk, M. (1995) “The Evolution of the SEI’s Capability Maturity Model for Software,” *Software Process: Improvements and Practice, 1*(1), 3-16.

[65] Pedhauzer, E. and Schmelkin, L. (1991) **Measurement, Design and Analysis**, Hillsdale, NJ: Lawrence Erlbaum Associates.

[66] Perry, D. and Schafer, W. (1995) “Editorial,” *Software Process: Improvements and Practice, 1*(1), 1.

[67] Pressman, R. (1996) “Software Process Perceptions,” *IEEE Software, 13*(6), 16-19.

[68] Orlikowski, W. (1993) "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development," *MIS Quarterly 18*(3), 309-340.

[69] Pondy, L. (1967) "Organizational Conflict: Concepts and Models," *Administrative Science*

Quarterly, 12, 296-320.
[70] Roach, S. (1991). Services under siege -- the restructuring imperative, *Harvard Business Review,* Sept-Oct., 82-92.
[71] Robey, D. (1984) “Conflict models for implementation research,” In *Applications of Management Science*, R. Schultz and M. Ginzberg, Eds. JAI Press: Greenwich, CT.
[72] Robey, D., Farrow, D. and Franz, C. (1989) 'Group Process and Conflict in Systems Development'. *Management Science, 35*(10), 1172-1191.
[73] Robey, D. and Newman, R. (1996). Sequential patterns in information systems development: An application of a social process model. *ACM Transactions on Information Systems*, *14*(1), 30-63.
[74] Sachs, P. (1995) “Transforming Work: Collaboration, Learning, and Design,” *Communications of the ACM*, 38(9), 36-46.
[75] Sawyer, S., Farber, J. and Spillers, R. (1997) “Supporting the Social Processes of Software Development,” *Information Technology & People, 10*(1), 46-62.
[76] Sawyer, S. and Guinan, P. (1995) "Application Development as Culture," *Proceedings of the 1995 AIS Americas Conference,* M. Ahuja, D. Galletta and H. Watson, (Eds), New York: ACM Press, 54-56.
[77] Seidler, J. (1974) "On Using Informants: A Technique for Collecting Quantitative Data and Controlling Measurement Error in Organization Analysis," *American Sociological Review, 39*(12), 816-831.
[78] Slocum, J., & Sims, H., Jr. (1980). A Typology for Integrating Technology, Organization, and Job Design. *Human Relations, 33*(3), 193-212.
[79] Steiner, I. (1972) **Group Process and Productivity**, New York: Academic Press.
[80] Stengel, R. (1997) “An Overtaxed IRS,” *Time, 67*(20), 58-62.
[81] Suchman, L. (1995) “Making Work Visible,” *Communications of the ACM*, 38(9), 56-65.
[82] Susskind, C. (1973) **Understanding Technology**, Baltimore: The Johns Hopkins University Press.
[83] Thomas, K. (1975) “Conflict and Conflict Management," M. Dunnette (Ed), **Handbook of Industrial Psychology**, Rand McNally, Chicago.
[84] Venkatraman, N. (1989) "The Concept of Fit in Strategy Research: Toward Verbal and Statistical Correspondence," *Academy of Management Review*, *14*(3), 423-444.
[85] Vessey, I and Sravanapudi, P., (1995) "CASE Tools as Collaborative Support Technologies," *Communications of the ACM, 38*(1), 83-95.
[86] Wald, M. (1996, November 25) “Future Hazy for Systems to Guide Ship Traffic,” *The New York Times*, C19.
[87] Walz, D., Elam. J. and Curtis, B. (1993) “The Dual Role of Conflict in Group Software Requirements and Design Activities. *Communications of the ACM, 36*(10), 63-76.
[88] Wasserman, A. (1996) “Toward A Discipline of Software Engineering,” *IEEE Software, 13*(6), 23-32.
[89] Weinberg, G. (1971) **The Psychology of Computer Programming**, New York: Van Nostrand Rheinhold.
[90] Wood, J. and Silver, D., (1989) **Joint Application Design**, New York: John Wiley & Sons.
[91] Yukl, G., **Leadership in Organizations**, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[92] Zachary, G. (1994) **Showstopper**, New York: Free Press.
[93] Zachary, G. (1998), “Armed Truce: Software in the Age of Teams,” *Information Technology & People, 11*(1), 59-66
[94] Zawacki, R. (1993) “Motivating IT People in the ‘90s: An Alarming Drop in Job Satisfaction,” *Software Practitioner, 3*(6), 1, 4-5.

NOTE 1: Scale Development

To develop the measured os social process we draw on two sources. For the ability of the team to manage intra-group conflict we use a scale developed and validated by [35]. To measure the level of informal coordination/ communication and feelings of supportiveness, we draw on a scale developed by Hackman [38] for use in evaluating group process behaviors. To depict the three factors used to assess the production processes, we draw on a measurement scale for technology that focuses on technology use [78] which is tailored for the software development domain [40]. The scale is adapted from [52].

We chose a set of performance measures that encompass a multi-attribute view of software performance. The three measures, product quality, team efficiency and team effectiveness are drawn from a scale used to assess software development team performance [41, 37, 36]. Self-reported perform-ance data were collected from the developers. This scale draws on the work of [37]. Scales were pre-tested prior to use and the entire scale was used to assess the factor for analysis (see the note in Table 1).

Please contact the first author for a copy of the instrument.

Appendix A: Factor Correlations, Means, and Standard Deviations

	Informal	Support.	Conflict	Formal	Method	Automated	Product	Team Performance	
	Coord.		Mgt.	Coord.	use	Tool use	Quality(SH)	(SH)	(SR)
Informal Coord.	4.58(.74)								
Supportiveness	.377**	4.01(1.10)							
Conflict Mgt.	.517***	.6139***	4.32(1.01)						
Formal Coord.	.460**	.571***	.693***	4.49(.88)					
Method use	.191	.117	.045	.162	4.63(.95)				
Autom. Tool use	.212	.066	-.070	.168	.511**	4.61(.97)			
SH-Prod. Qual.	.053	.367**	.416**	.411**	.065	.182	5.16(.70)		
SH-Team Perf.	.070	.353*	.286	.448**	.160	.304	.705***	5.09(.75)	
SR-Team Perf.	.572***	.500***	.624***	.662***	.093	.309	.371**	.420**	4.40(.74)

Notes:
(1) diagonal is mean (standard deviation) based on seven-point scale with 1 low/bad and 7 high/good
(2) * = $p < 0.05$; ** = $p < 0.01$, *** = $p < 0.001$, n =40 teams
(3) The matrix represents the zero order correlation based on Pearson product moment, two tailed. A significant correlation indicates a relationship but not any causality. For example, the significant correlation between formal coordination and stakeholder rated product quality means that as one changes, so will the other. We *imply* the causality be theorizing that higher levels of formal coordination lead to higher levels of product quality.

Table 1: Factors and Scale Questions

Factor (scale reliability)[1]	**Scale questions (defining the factor)**
Informal Communication and Coordination (.70):	The amount of communication with team-members. The amount of communication with key people who are not part of the team. The amount of coordination with team-members. The amount of coordination with key people who are not part of the team.
Conflict Management (.71):	The ability to surfacing differences among the group. The ability of the group to negotiating acceptable shared agreements. The ability of the team members to reach compromises.
Supportiveness (.71):	Team-members support other members of their team Team-members are loyal to the team. Team-members try to cover for each other as needed.
Formal Coordination (.82):	The use of scheduled meetings. The value of scheduled meetings. The use of project management documents. The value of project management documents. The level of information sharing required as part of development methodology. The value of information sharing required as part of development methodology.
Method Use (.88):	The use of prescribed methods or patterns. The value of prescribed methods or patterns. The use of formal project reviews. The value of formal project reviews. The use of required documentation in developing software. The value of required documentation in developing software.
Automated Tool Use (.71):	The use of automated development software. The value of automated development software. The use of shared code repositories/libraries. The value of share code repositories/libraries. The use of tools for common access to work products when developing software. The value of tools for common access to work products when developing software.
Stakeholder-rated Product Quality: (.91)	Quality of the system produced by the project team. Number of defects in the system.

	Extent to which the system adds value to our firm.
	The extent to which the system adheres to organizational standards.
Stakeholder-rated Team Performance (.91):	Efficiency of project team operations.
	Adherence to schedules during the project.
	Amount of work the project team produced.
	Ability of the project team to meet the goals of the project.
	Extent to which the users' business needs are reflected in the system.
	The contribution of the system to the performance of the firm.
Self-reported Team Performance (.95):	Efficiency of project team operations.
	Adherence to schedules during the project.
	Amount of work the project team produced.
	Ability of the project team to meet the goals of the project.
	Extent to which the users' business needs are reflected in the system.
	The contribution of the system to the performance of the firm.

Note 1: Factor reliability is a measure of the degree to which responses to these questions co-vary. This is measured using Chronbach’s alpha. If the alpha is greater than 0.70, the scale comprising a set of indicators is considered reliable for exploratory work. Several scales are much higher, suggesting that the use of previously validated scales improves the value of these factors.

Table 2: Team and Individual Characteristics (1)

	Years (mean)
Individual professional experience:	12.0
Individual as an employee:	4.2
Individual management experience (2):	4.5
Individual in present position:	2.3
Individual as a team member:	1.5
Team with 75% of present Team membership:	1.9
Team with same Team Leader:	1.0

	Number (mean)
Individual Project experience:	9.8
Team Size	9.0

	Education (%)
College Degree:	89%
Masters/Ph.D.:	34%

(1) Not allowed to collect age and gender data.

(2) For those having previous management experience and are not presently managing. Reflects responses from 27% of sample.

Table 3: Effects of Methodology Factors to Performance

Factor	Stakeholder-rated Product Quality	Stakeholder-rated Team Performance	Self-reported Team Performance
Method Use	No contribution	No contribution	No contribution
Automated Tool Use	No contribution	No contribution	No contribution
Variance explained (%) (adjusted r^2)	Non Significance	No Significance	No Significance

where: * = $p < 0.05$; ** = $p < 0.01$, *** = $p < 0.001$, n =40 teams

Table 4: Effects of Methodology and Formal Coordination Factors to Performance

Factor	Stakeholder-rated Product Quality	Stakeholder-rated Team Performance	Self-reported Team Performance
Method Use	No contribution	No contribution	No contribution
Automated Tool Use	No contribution	No contribution	No contribution
Formal Coordination	No contribution	.495**	.679***
Variance explained (%) (adjusted r^2)	N.S.	22.2**	50.0***

where: * = $p < 0.05$; ** = $p < 0.01$, *** = $p < 0.001$, n =40 teams

Table 5: Effect of Production and Social Process Factors to Performance

Factor	Stakeholder-rated Product Quality	Stakeholder-rated Team Performance	Self-reported Team Performance
Method Use	No contribution	No contribution	No contribution
Automated Tool Use	No contribution	No contribution	No contribution
Formal Coordination	.298*	.409**	.544***
Informal Coord./Comm.	.321*	.293*	.575***
Conflict Management	.395**	.399**	.571***
Supportiveness	No contribution	No contribution	.611***
Variance explained (%) (adjusted r^2)	19.0*	23.1***	69.0***

where: * = $p < 0.05$; ** = $p < 0.01$, *** = $p < 0.001$, n =40 teams

Table 6: Moderating Effects of Low Levels of Methodology to Performance

Factor	Stakeholder-rated Product Quality	Stakeholder-rated Team Performance	Self-reported Team Performance
Formal Coordination	.541***	No contribution	.400**
Informal Coord/Comm.	.294*	No contribution	.371**
Conflict Management	.324**	No contribution	
Supportiveness	No contribution	No contribution	No contribution
Variance explained (%) (adjusted r^2)	25.1*	No Significance	62.0***

where: * = $p < 0.05$; ** = $p < 0.01$, *** = $p < 0.001$, n =20 teams

Figure 7: The Use, and Value of, Methods and Automated Tools

(Plan to do means comparison in bar chart format)

For each question, respondents asked to first evaluate team use, then value to the team, using seven-point Likert scales. For the AUSE@ query, the A1" meant ANot Used@ and A7" meant AUsed a lot.@ For the AVALUE@ query, the A1" meant ALow@ and A7" meant AHigh.@ The value A4" was ANeutral@ for both questions.

Method Use

Prescribed Methods or Patterns

Use	4.4
Value	4.8

Formal Project Reviews

Use	4.6
Value	5.0

Required Documentation

Use	4.8
Value	5.0

Automated Tool Use

Automated Development Software

Use	3.2
Value	3.8

Shared code repositories/libraries

Use	5.2
Value	5.6

Tools for common access to work products

Use	5.4
Value	6.0