2000

# Software Process Management: An Organizational Learning Perspective

T. Ravichandran
*Rensselaer Polytechnic Institute*, ravit@rpi.edu

search

# Software Process Management: An Organizational Learning Perspective

T. Ravichandran, Lally School of Management & Technology,
Rensselaer Polytechnic Institute, Troy, NY 12180, E-Mail: ravit@rpi.edu
Arun Rai, J.Mack Robinson College of Business, Georgia State University, Atlanta, GA 3-303

*Abstract*-**We draw from the quality management and organizational learning literatures to develop a descriptive model of software process management. These literature streams suggest that the concepts of process design, process control, learning through experimentation and learning through knowledge codification and reuse are important in the development of process capability. We define each of these concepts as latent constructs, and then propose a descriptive model of software process management. The model was tested using data collected from 123 IS units in Fortune 1000 firms and large government agencies. Our empirical results suggest that process design efficacy and process control have a positive relationship with software process capability. Furthermore, both learning through experimentation and learning through knowledge codification and reuse enable the design of efficacious processes and facilitate the development of appropriate standards to control the development process. Implications of our findings for IS research and practice are discussed.**

## I. Introduction

The history of systems development in organizations has been marked by performance shortfalls including abandoned projects, delivery of systems that do not meet user needs and expensive project failures. It is estimated that only 16% of the systems development projects undertaken by Information Systems (IS) departments are completed on time and within budget [21]. Even among the successfully completed projects, the delivered systems fail to meet user requirements and are of poor quality. These problems are compounded by the escalating demand for new systems and because systems development is in the critical path to getting new products or services to market [18].

Recommendations to improve systems development performance have focused on a variety of factors, including the use of modern systems design methods (such as prototyping and rapid application design), use of technologies such as object-oriented computing, project management practices, and approaches that consider the underlying process used across projects. While each of these approaches is important, this study focuses on software process management because of the considerable attention that process-based approaches have received recently in industry and academia. Important examples of such approaches include the capability maturity model (CMM) of the Software Engineering Institute and total quality management (TQM). A basic premise of these process-based approaches is that the capability of the systems development process largely determines systems delivery outcomes.

Several process improvement models have been proposed by the software engineering community, the most popular of which is the CMM. These models are increasingly being adopted by IS units and have been effective in establishing process management as an important strategy for systems delivery performance improvement. Anecdotal evidence suggests that organizations implementing software process management have realized gains in development cycle time and programmer productivity [9,11,13]. Reports also suggest that organizations face difficulties in implementing the prescriptions of the software process improvement models such as the CMM [4,17,19].

The lack of theory informing existing software process improvement models inhibits development of meaningful generalizations based on the experience of the firms implementing software process management. Existing software process improvement models are *normative* models that stipulate best practices for improving systems development performance. These models provide a wide variety of tools, techniques and management practices aimed at structuring and controlling the development process. While effective in reconfiguring existing knowledge to make it more useful for practice, normative process improvement models may not provide a fundamental understanding of what process management is and how it impacts systems delivery performance. Our survey of the IS and software engineering literatures revealed that very limited theoretical work has been done in the area of software process management. Given the growing interest in academia and among IS managers in software process management, theory development in this area is very much in need.

In this paper, we develop a *descriptive* model of software process management. In contrast to normative models, *descriptive* models focus on understanding a phenomenon, identifying its dimensions, developing associations between those dimensions, and developing generalizations about those associations. Research in this tradition seeks to define research constructs, develop scales for them, and examine cause-effect relationships in an effort to develop theoretical knowledge. The theoretical starting point for our research is rooted in the fundamental TQM percept that learning and knowledge creation are central to improving process capability. The TQM literature characterizes process management as a continuous effort to design an efficacious process by understanding the relationship between process configurations and process outcomes and embedding the knowledge in the process

through routines and formal process definitions. Synthesizing prior research in TQM and software engineering, we identify four dimensions of software process management: process design, learning through experimentation, learning through knowledge codification and reuse, and process control. Process design refers to how the development process is configured; learning through experimentation refers to on-going attempts to understand cause-effect relationships between process parameters and process outcomes; learning through knowledge codification and reuse refers to the extent to which procedural and declarative knowledge are codified for future use; process control refers to the extent to which the development process is controlled through standards. We develop a research model to examine, (1) the interrelationships between these dimensions of process management and, (2) the relationships between the dimensions of process management, and process capability. We test our research model using data collected from 123 IS units in Fortune 1000 firms and large government agencies.

## II. Software Process Models: A Review and Critique

Software process improvement has emerged as an important paradigm for managing software development in organizations. This is evident from the variety of models that have been put forth in the past few years and the increasing attention they are receiving from academics and practitioners. Models such as the CMM and BOOTSTRAP provide guidelines for improving process capability and present standards and instruments for process maturity assessment. On the other hand, frameworks such as the ISO 9000 are primarily intended to assess process capability. Other models such as the TRILLIUM have been developed for software development processes in specific industries such as telecommunications. Since there are significant similarities between the various software process models, we will review only the CMM here. For a good overview of the key software process models, the readers are referred to Zharan [24].

The CMM provides a framework for evaluating, assessing and improving process capability. The model defines an evolutionary path from ad hoc , chaotic processes to mature, disciplined processes and stipulates key practices that reflect the capability of the development process. Process capability, as assessed by the predictability of development outcomes in terms of budget, schedules, and quality, is enhanced when feedback is meaningfully generated and utilized to recalibrate and fine tune process design. The model identifies key process areas that have to be improved in order to enhance process capability and for each process area, it stipulates key management practices that reflect a capable process.

CMM is now popular and has been effective in emphasizing the importance of process management among IS practitioners. Anecdotal evidence suggests that organizations implementing CMM-based software process improvement have realized gains in development cycle time and programmer productivity [9,11,13]. Reports also suggest that organizations face difficulties in adhering to the normative sequence as recommended by CMM, in which changes to the development process need to be implemented [4,17,19]. Other problems with the model include the low priority accorded to technology issues, the inapplicability of some maturity assessment criteria to different software development contexts, and the use of a fixed template with a sparse set of questions to analyze complex systems of software processes [3].

The lack of theory informing the conceptualization of the CMM stages has been pointed out as a major shortcoming of the model. While intuitively appealing, the CMM maturity levels lack theoretical and empirical support. Moreover, the role of learning in the development of process capability has not been clearly addressed in the CMM. For example, it is unclear how the various practices recommended by the CMM promote learning and, the knowledge accrued gets embedded in the development process. In contrast, much of the TQM and continuous improvement literatures have framed process improvement as a systematic attempt to foster learning. In this study, we conceptualize software process management as an organized effort to improve software development performance through learning and knowledge sharing.

## III. An Organizational Learning Perspective of Software Process Management

Process management has a long history that can be traced back to statistical process control theories in the quality management literature. From these origins, process management methods have blossomed to include techniques aimed at better management and control of product and service design, and scientific methods of continuous improvement, such as the plan-do-check-act cycle. The methodological approaches underlying process management enable progressive refinement of process design based on continuous learning and knowledge sharing. Implicit in the process management philosophy is the awareness and institutionalization of process thinking. Process thinking differs from traditional goal-oriented management in the belief that simultaneous improvements to multiple performance objectives, such as cost, quality and productivity, can only be achieved by improving process quality. Organizations that embrace the former view focus on process capability and consider product quality to be a consequence of a quality process, while organizations that buy-in to the later view consider product quality, in and of itself, as a strategic goal. For example, Deming [8] differentiated between zero defects as a process versus a company goal. He explained that both approaches lead to zero defects. Whereas zero defects as a company goal could lead to its achievement at the price of inspection and dismal productivity, improvements in the process can lead to zero defects as a natural consequence.

Process thinking is yet to be institutionalized in most IS organizations. IS organizations continue to rely more on testing and debugging as the primary means to improve systems quality despite the inefficiencies and high cost of trapping and fixing bugs during the testing stage. A more effective approach would be to focus on improving

the capability of the development process so that quality is engineered into the system in the early development phases, such as requirement analysis and design. Use of quality function deployment to better translate user requirements into systems design [25], reuse of well-tested design models and primitives from prior development projects, and use of formal design methods are some illustrations of the shift from an outcome focus to a process focus.

A fundamental percept of process management is that systematic learning is required to continuously improve process capabilities. The goal is to improve the process design by understanding the cause-effect relationships between process parameters and process outcomes. This is achieved through the extraction, synthesis and analysis of information presented by process variations. According to Deming [8], product, process and service variations are indicative of the lack of deep knowledge about the special and common causes of variations, the understanding of which is necessary to improve process capability. Process improvement through variance reduction corresponds well with Argyris & Schon's [2] theoretical work on single and double loop learning [1]. The control mechanisms integral to process management emphasize routines and performance standards which promote first-order learning as they encourage sustained attempts to maintain or exceed desired performance goals. The resultant task knowledge facilitates achievement of performance objectives within the constraints of a defined work process. On the other hand, systematic collection and use of process information can facilitate identification of the causes of process variations and thus could lead to fundamental changes to the process. This represents second order learning.

The process management practices are also grounded in the literature on learning curves [6,10,16]. These authors have proposed that extended production experience provide the employees an opportunity for learning that may lead to predictable performance improvements. A more sophisticated conceptualization of the learning curve effects is implicit in process management where incremental process changes are planned, tested, observed and appropriately implemented in order to improve performance [1]. These experiments involve the four aspects of learning described by Huber [14]: information acquisition, information interpretation, information distribution and organizational memory. Once a process change is made, information about the impact of the change on product and process variations, quality performance and process characteristics must be collected. This information must then be systematically analyzed and interpreted to understand root causes of quality problems and process variations. Finally, the knowledge gained through such information analysis should be embedded in the process through appropriate changes in process design. Such institutionalization of process knowledge as formalized work methods and routines is critical to ensure that process knowledge is preserved in the organization.

However, continuous improvement based on incremental change is inadequate in contexts such as systems development where the process is not completely standardized and could vary across development projects. Unlike manufacturing processes, where knowledge accrued through repetition of the same process can be a basis for process design and control, software process management must be based on something less definitive - the ability to learn from other software development projects. In addition to facilitating learning through experimentation, one of the goals of software process management should be to create the infrastructure to facilitate encoding, transfer and reuse of knowledge assets across projects.

The organizational learning literature differentiates between declarative knowledge and procedural knowledge but posits that reuse of both types of knowledge is important from a performance improvement standpoint. Declarative knowledge represents facts and factual relationships and is also referred to as *know-what*. It provides definitions of elements, and interrelationships between elements. In the context of systems development, declarative knowledge could be knowledge about the business domain that is encapsulated in design models, knowledge about data structures, metadata, and business logic in the form of programs. Once validated, these forms of declarative knowledge can be reused in subsequent projects.

Procedural knowledge pertains to knowledge about the process of transforming inputs into useful outcomes. Procedural knowledge has been referred to as *know-how* and is oriented to establish a meaningful course of action for a given situation. In the context of systems development, this pertains to the process of transforming inputs, such as user requirements, technology, and skills, into functional application systems. The methodologies used in systems development are an encapsulation of procedural knowledge. The project management practices and routines also embed critical procedural knowledge. Standardizing the design and development methods used across projects and sharing project management experiences through common project databases are some means to reuse procedural knowledge.

In summary, from an organizational learning perspective, software process management should include four critical aspects. First, an efficacious development process must be designed that reflects the state-of-the-art in tools, techniques, methods and procedures for systems development. Second, mechanisms to promote learning by sharing and reuse of procedural and declarative knowledge must be established. Third, mechanisms for learning, through systematic analysis of system quality problems and identification of their root causes should be implemented. Such analyses are likely to provide valuable process knowledge that could be used to refine the development process and, to evolve process and outcome standards. Fourth, these evolved performance standards should be used to control the development process in order to ensure that process outcomes are achieved in an efficient manner. Furthermore, these standards should be refined periodically to reflect the changes in the development process.

## IV. Research Model and Hypotheses

The theoretical model for this study is presented in Figure 1. Broadly, the model depicts that software process management has a positive impact on process capability. More specifically, the model depicts the four constructs of software process management, their interrelationships and how they impact software process capability. We argue that the design of the development process and the control systems in place directly impact the capability of the development process. It is logical to expect a development process that is designed to reflect the state-of-the-art in methods, tools and techniques to be more capable than one that is not. Moreover, appropriate control systems are required to ensure efficient task execution. Thus, both process design and process control are posited to have a direct positive impact on process capability. On the other hand, learning through experimentation and knowledge codification and reuse are posited to have an indirect effect on process capability by enabling the design of an efficacious development process and the development of appropriate process control standards. The rationale behind these relationships is the notion that knowledge resources can yield tangible outcomes only if they are put to use.

### *Process Capability*

Process capability indicates the richness of an organization's software development process and reflects the consistency with which process outputs can be produced. We used the process maturity levels as defined in the CMM to assess the capability of the systems development process (Table 1). The CMM defines five process maturity levels starting from "initial," which represents a chaotic, undisciplined process to "optimized," which represents a sophisticated development process that is being continuous refined. Lower process maturity levels are indicative of a state where system quality, cost, and leadtime are unpredictable. Higher maturity levels on the other hand, are indicative of a state where the process outputs could be controlled by varying process parameters and that the process is being continuously refined.

### *Process Design*

Process design refers to how the systems development process is configured. Efficacious process design reflects the use of state-of-the-art tools, techniques, methods and accumulated knowledge about systems development. Based on a review of the IS and software engineering literatures, we identified four key indicators of efficacious process design: process automation, process integration, formalization of design methods and formalization of reusability. *Process automation* refers to the extent to which hardware and software tools are deployed to support systems development tasks. There is extensive empirical evidence in the software engineering literature that automation tools enhance the capability of the development process. Automation of back-end tasks, such as coding and debugging, can improve process efficiency, program quality and documentation quality and, provide a degree of control over these tasks. Automation support for front-end tasks, such as requirement assessment and design, facilitate exchange of information, models and design, and thus reduce errors due to poor interfacing between systems development tasks. They also provide an effective means to abstract domain requirements into understandable forms (such as models, diagrams, relations) and facilitate their manipulation and exchange among users and developers [12].

*Process integration* refers to the extent to which tasks across different systems development phases, such as planning, analysis, logical design , physical design and construction are integrated with each other. Integration could lead to significant process gains by eliminating mediating tasks involved in translating requirements into design and design into code. Furthermore, the ease of propagating changes in domain requirements to design and code in an integrated development process provides an effective means to react to changes in user requirements.

*Formalization of analysis and design methods* pertains to the extent to which adherence to standard systems design techniques and methods is integral to the systems development process. A large proportion of system quality problems has been attributed to requirements assessment and design [23]. Hence, practices that reduce or eliminate quality problems due to design weaknesses are critical aspects of an efficacious systems development process. Formalization of analysis and design methods to focus attention on customer needs and develop complete and accurate requirements is an important property of an efficacious systems development process.

*Formalization of reusability* pertains to the extent to which reuse is encouraged and enforced as part of ongoing systems development tasks. An important theme underlying process design is waste elimination and error prevention, as opposed to error detection. Design and code modules that have been effectively developed and tested for other application systems can often be deployed elsewhere in similar application development contexts. Such a strategy is oriented to reduce duplication, waste and introduction of unnecessary errors in the development process. Thus, formalization of reusability in systems development is recognized as an important property of an efficacious systems development process.

In summary, process automation, process integration, formalization of design methods and formalization of reusability are important features of a development process. The extent to which these process features are implemented is likely to determine the overall capability of the development process. The notion that the design of the process determines it capability is well supported by the CMM framework where several key process areas are identified for assessment at each maturity level. Thus, the following relationship is proposed:

*H1: There is a positive association between efficacious process design and systems development process capability.*

### *Process Control*

Process control refers to how the development process is monitored to ensure that activities are performed according to process definitions and that the process goals are achieved. An effective means to exercise control is through performance standards. Both *outcome standards and process standards* are required to effectively control the development process. Outcome standards delineate targeted performance levels for systems development tasks (such as design, programming, documentation). Outcome control is essential to ensure that performance improvement efforts result in tangible outcomes. Schaffer & Thompson [20] found that process improvement efforts that are not accompanied by tight controls have not been very successful as they result in excessive exploration with very little exploitation of accrued knowledge.

Process standards in the form of development methodologies and procedures provide a means to put to use the knowledge embedded in the development process to attain process goals in a predictable manner. As pointed out earlier, adherence to a standard development process is necessary for cumulative learning to occur across different development projects. Thus,

*H2: There is a positive association between process control and systems development process capability.*

### *Learning through Experimentation*

Learning through experimentation is largely an effort to develop deep process knowledge through controlled changes to the development process. Deep process knowledge includes an understanding of the cause-effect relationships between process parameters and process outcomes. Deming [8] argued that analysis of variations in product quality to understand their root causes is critical to develop deep process knowledge. Thus, the extent to which *quality data is collected and used* is reflective of learning through experimentation.

Weick [22] pointed out that often inappropriate or bad practices get institutionalized in organizations partly due to lack of validated procedural knowledge. He also pointed out that, once institutionalized, these routines will persist even after those involved in developing those routines have left the organizations. Development of process knowledge is easier in situations where activities are sufficiently routine to be well understood. Furthermore, in such contexts, procedural knowledge can be easily validated and inappropriate routines and process design may be corrected. On the other hand, in fundamentally uncertain contexts, such as in systems development, development of process knowledge rests on the organization's ability to validate its procedural knowledge through controlled experiments and benchmarking. In addition, knowledge from external sources must be effectively integrated with those generated internally to develop critical process knowledge. Clearly, process design that is not driven by such systematic learning is more likely to be ineffective. Similarly, process control standards must be based on validated process knowledge for them to be reasonable and yet serve as a catalyst for process improvement. Thus,

*H3: There is a positive association between learning through experimentation and efficacious process design.*

*H4: There is a positive association between learning through experimentation and process control.*

### *Learning through Knowledge Codification and Reuse*

Knowledge codification and reuse refers to the creation of knowledge assets for use across multiple projects. Since the repository is essential for sharing knowledge assets, its use is an indicator of the extent to which knowledge generated in each project is codified and packaged for reuse. As mentioned earlier, significant process gains can be realized by reusing declarative knowledge generated in each project. Declarative knowledge is encapsulated in various software assets such as design models, code and documentation. Repositories also enable the sharing of procedural knowledge. Project databases provide access to codified information about past projects and templates and standards for project management tasks such as cost estimation, project planning and scheduling. In addition, lessons learned in past projects can be summarized and disseminated.

Reuse of software assets and project experience are critical in moving from a craft approach where performance is largely a function of individual skills, to a factory approach where the process plays a dominant role in determining systems delivery performance [7]. Thus,

*H5: There is a positive association between knowledge codification and reuse and efficacious process design*

*H6: There is a positive association between knowledge codification and reuse and process control.*

## V. DATA COLLECTION

Data to test our model was collected as part of a larger study focused on quality management in systems development. A national survey of IS units in Fortune 1000 firms and large government agencies in the U.S was conducted to collect data for the study. Senior IS executives were chosen as the respondents as they are likely to be most informed about quality initiatives in IS units. A total of 710 questionnaires were mailed. A total of four mailings, each spaced apart by three weeks, were undertaken. 123 usable responses were received resulting in a response rate of 17.32%. A variety of tests were performed to test for nonresponse bias. These tests revealed no systematic bias suggesting that the respondents can be pooled with no loss in generalizability. Our sample represents a broad cross-section in terms of industry, organization size and IS department size.

### *Measures*

The CMM process maturity framework was used to measure systems development process capability. We used an unidimensional response matrix enumerating the

five maturity levels along with their descriptions to measure process maturity. The descriptions of the maturity levels were borrowed directly from those given in the Capability Maturity Model [15]. Table 1 indicates the distribution of the process maturity levels across the organizations sampled. We compared the distribution of the self-reported process maturity levels with the results of diagnostic surveys conducted by SEI. The chi-square analysis revealed no significant differences.

Process automation was measured by assessing the number of systems development tasks, such as requirements analysis, systems design, code generation, metrics collection, testing and project management, for which automation support was provided. Process integration was measured using the following 5-item scale: 1-Unintegrated, 2-Tasks are integrated within individual development phases, 3-Tasks are integrated across a few development phases, 4-Tasks are integrated across multiple but not all development phases, 5-tasks are fully integrated across all development phases.

Multi-item scales were used to measure each of the other constructs in the model. The items for each scale were identified based on a review and synthesis of the TQM, IS and software engineering literatures. The reliability and validity of these scales were systematically assessed using standard procedures for scale validation. Table 3 depicts the scale items, factor loadings, eigen value and the Cronbach's alpha value of each scale. It is seen from the table that all the scales have acceptable levels of reliability and convergent validity (details of scale validation are not presented to contain the length of the paper).

### *Statistical Analysis and Results*

We used the partial least square (PLS) method of structural modeling to test the research model. In order to estimate the significance of path coefficients, weights and loadings a bootstraping technique was used to generate 200 samples. Weights indicate the relative importance of the indicators in defining the formative constructs. For formative indicators, which have a regression-like relationship with the latent construct only, the weights (and not the loading) need to be considered in assessing the measurement model [5]. While no minimum threshold values for indicator weights have been established , the statistical significance of the weights can be used to determine the relative importance of the indicators in forming a latent variable. All except one, indicator weight are statistically significant. Specifically, the weight for process integration was not statistically significant ( $b = .130; p < .082$). We dropped this variable and reassessed the model. All indicator weights were statistically significant in the revised model. The results (Figure 2) indicate that 40% of the variance in process design, 58.2% of the variance in process control and 15.1% of the variance in process capability were explained by the revised model. Overall, these results provide adequate support for our model and the associated hypotheses.

## VI. DISCUSSION

Our objective was to develop a descriptive model of software process management. We identified four key dimensions of software process management and examined how they interrelate to impact process capability. The results support our model for software process management. All path coefficients in the model are statistically significant and their values are indicative of the positive relationships between the constructs in the model.

Process capability needs to be developed by engineering both process design and control systems. Our empirical results suggest that process control has a marginally stronger impact on process capability than process design. While IS units have focussed their attention on development process design and adoption of new development tools and technologies, they have paid limited attention to setting standards for controlling the process. Consequently, IS units often do not realize significant performance improvements as they do not fine-tune their processes by using well-designed control systems. Our results suggest that utilization of knowledge embedded in the development process, through consistent efforts to meet meaningfully established performance standards is as important as periodically redesigning the process.

We found that both learning through experimentation and learning through knowledge codification and reuse have a strong positive effect on process design. While it appears intuitive that efforts to improve the development process should be based on a good understanding of the relationships between process parameters and process outcomes, it often may not be the case due to several reasons. Development of process knowledge is a fairly complex endeavor even for simple tasks because of the possible variations in input and process parameters that can affect outputs. The challenges are even higher in the context of systems development due to task complexity and the constant need to adapt the process to technological and business changes. Thus, an IS unit' s ability to learn is a critical meta-capability that needs to be nurtured. We examined two important learning mechanisms that have been emphasized in the TQM and organizational learning literatures. Future research should examine the role of other learning mechanisms in software process improvement.

Finally, we found that both learning through experimentation and learning through reuse of past experiences have a strong positive effect on process control. In standard-maintaining organizations, control systems tend to be conceptualized as static and non-evolutionary. Such organizations tend to have static process capabilities that perform at a defined capability level. On the other hand, continuous improvement organizations use dynamic standards, which are systematically ratcheted up. Learning, adaptation and sharing declarative and procedural knowledge about the development process are important for the evolution of meaningful standards. Process databases should be analyzed to determine constraints that need to be alleviated so as to reach higher levels of process capability. Low standards should be identified and questioned, while standards that are not

being attained can be critically examined. Contradictory standards, which may be in use at different stages in the development process, should be surfaced, as should inconsistencies between process and outcome standards.

Software process models, such as the CMM, have become popular. However, reports suggest that organizations face difficulties in adhering to CMM's normative sequence in which changes to the development process need to be implemented [4,17,19]. We focused on developing a theoretical understanding of the dynamics of process management. The substantive results of our study complement existing normative software process improvement models. Software process improvement initiatives often do not succeed, but as yet there is little theory to explain the differences between successful and unsuccessful efforts. Models such as the one developed here provide a framework to incorporate accumulated knowledge about process management in other disciplines with those in software engineering. Such integrative efforts are required to develop a deeper understanding of software process management.

TABLE 1
DISTRIBUTION OF PROCESS MATURITY LEVELS

| Value | Frequency | Percentage | Cumulative Percentage |
|---|---|---|---|
| Initial | 5 | 4.2 | 4.2 |
| Repeatable | 25 | 20.8 | 25.0 |
| Defined | 65 | 54.2 | 79.9 |
| Managed | 17 | 14.2 | 94.3 |
| Optimized | 4 | 3.3 | 97.6 |
| Missing | 4 | 3.3 | 100.0 |

TABLE 2
FACTOR STRUCTURE AND RELIABILITY OF MULTI-ITEM SCALES

| Factors/ Items | Loading |
|---|---|
| **Formalization of Design Methods**<br>**Eigen Value: 2.373; Variance Extracted: 59.5%; Cronbach's** | **.77** |
| Formal techniques such as JAD and prototyping are regularly used for requirement elicitation | .84 |
| Idea generation techniques such as brain storming are used in system design | .71 |
| Formal techniques such as quality function deployment are used to translate user requirements into design | .76 |
| Standard representation schemes such as ER diagrams and DFD are used for design specifications | .77 |
| **Formalization of Reusability**<br>**Eigen Value: 2.807; Variance Extracted: 70.2%; Cronbach's** | **.85** |
| Formal policies to promote development of reusable design/code have been implemented | .85 |
| Formal policies that mandate use of reusable components have been implemented | .92 |
| Reuse of code/design components is monitored | .87 |
| Formal policies on parameterization of design/code has been implemented | .70 |
| **Control Through Outcome Standards**<br>**Eigen Value: 4.147; Variance Extracted: 82.9; Cronbach's** | **.95** |
| Performance standards have been established for design | .94 |
| Performance standards have been established for programming | .96 |
| Performance standards have been established for testing | .94 |
| Performance standards are used to monitor and control output | .89 |
| Performance standards are revised annually/regularly | .82 |
| **Control Through Process Standards**<br>**Eigen Value: 1.478; Variance Extracted: 73.9%; Cronbach's** | **.65** |
| Standard procedures for systems development are strictly adhered to | .86 |
| Vendors/consultants are required to adhere to a standard development methodology | .86 |
| **Information Collection and Use**<br>**Eigen Value: 4.228; Variance Extracted: 52.9%; Cronbach's** | **.87** |
| Quality data is collected and reported at frequent intervals | .80 |
| Vendors/consultants are pressed to furnish quality data | .56 |
| Performance levels are benchmarked with those of other firms | .63 |
| Quality problems are analyzed to identify problem causes | .77 |
| Quality data is systematically used in managing systems development | .88 |
| Cost of quality is analyzed | .71 |
| Metrics are recaliberated to reflect changes in the development process | .76 |
| Best practices are systematically institutionalized | .65 |
| **Use of Repository**<br>**Eigen Value: 2.661; Variance Extracted: 53.2%; Cronbach's** | **.77** |
| Standardized system interface and access to program libraries have been implemented | .82 |
| Centralized library system to store programs and models have been implemented | .64 |
| Development database for project management have been implemented | .73 |
| Databases across interdependent organizational units are integrated | .77 |
| Common data definitions sharable across applications have been developed | .69 |

References available upon request from the first author .

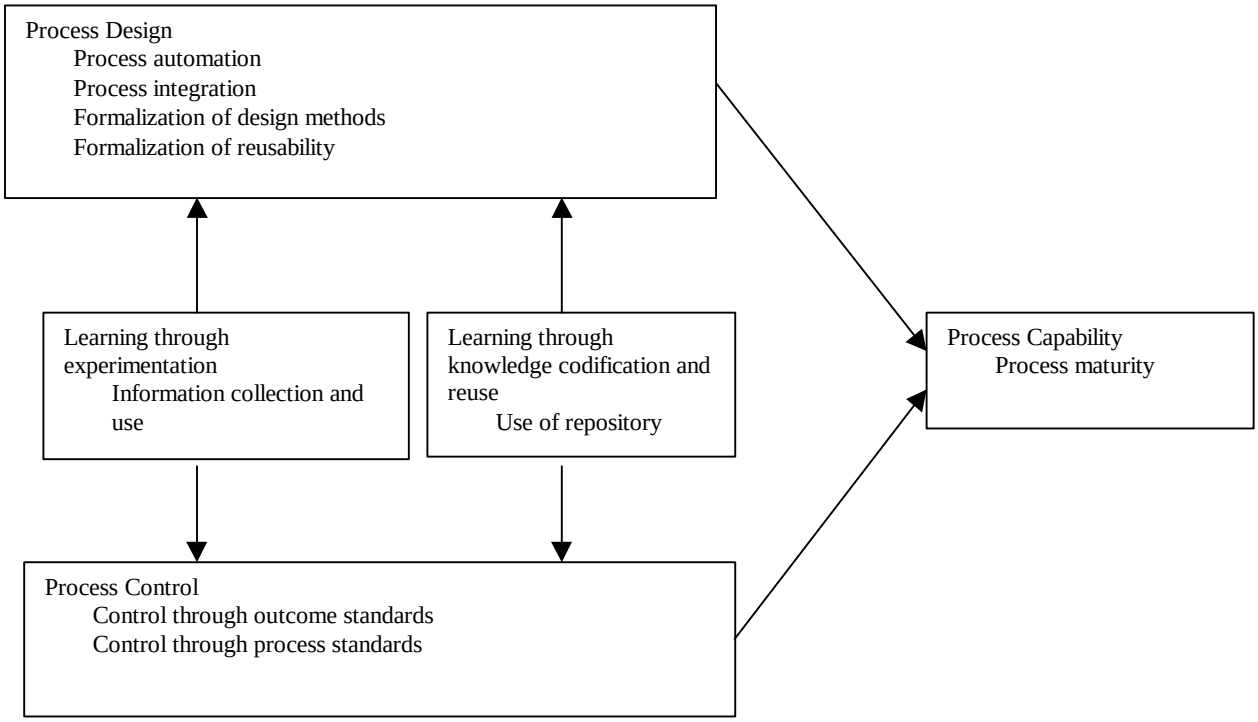


Fig. 1. Conceptual Model

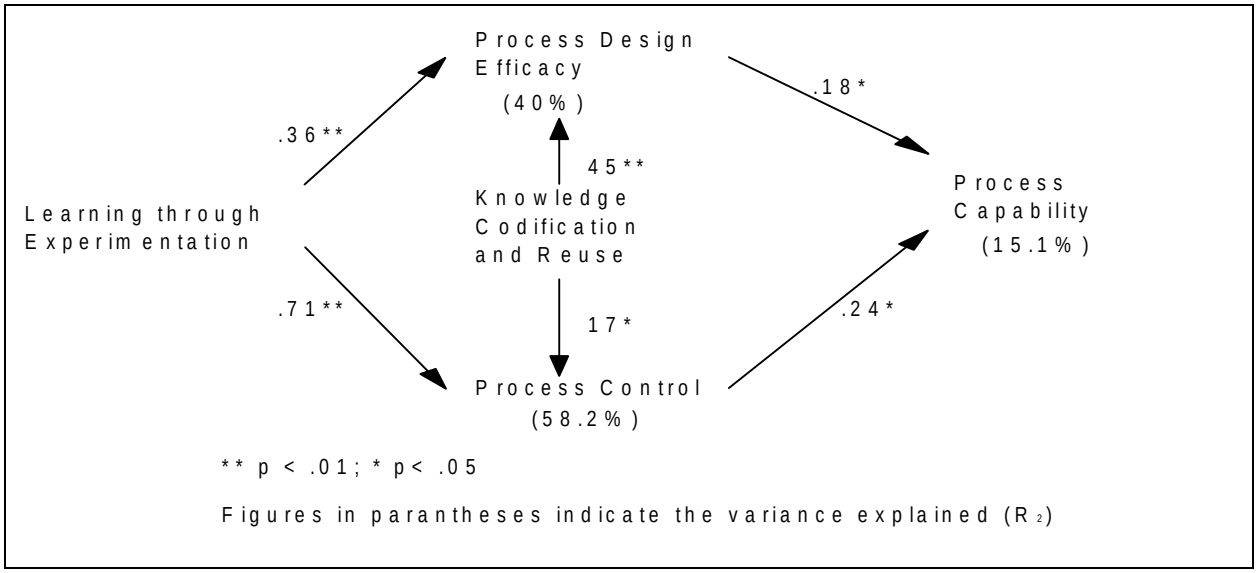


Fig. 2. Significant Paths and Path Coefficients in the PLS Model