



ELSEVIER

Available online at www.sciencedirect.com



Information and Software Technology 46 (2004) 899–906

**INFORMATION
AND
SOFTWARE
TECHNOLOGY**

www.elsevier.com/locate/infosof

An empirical study of the effect of knowledge integration on software development performance

Amrit Tiwana*

Goizueta Business School, Emory University, 1300 Clifton Road, Atlanta, GA 30322, USA

Received 26 December 2003; revised 18 March 2004; accepted 27 March 2004

Available online 1 June 2004

Abstract

Although the role of integrating application domain knowledge with technical knowledge is implicitly recognized in software engineering practice, no large scale study has attempted to validate this empirically in a field setting. In this paper, a large-scale empirical study of 232 software development projects in 232 software development organizations shows that higher integration of business application domain knowledge with technical knowledge during the software development process increases software development effectiveness, reduces defect density throughout the development trajectory, lowers warranty defects, and increases software development efficiency. The findings highlight the influence of knowledge integration on various dimensions of software development performance.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Knowledge management; Knowledge integration; Software development performance; Software quality

1. Introduction

Integration of technical knowledge with business application domain knowledge is central to effective software development. However, no empirical study to date has assessed how and to what extent does knowledge integration affect various facets of software development performance. This paper develops and empirically validates a multi-dimensional model showing how knowledge integration influences software development performance. The model is tested using field data on 232 globally distributed software projects in 232 software development companies. The results demonstrate that higher levels of knowledge integration enhance design effectiveness, lower defect density, and increase software development efficiency. Its relative influence on each of these performance dimensions is also empirically assessed.

1.1. Research questions

Two research questions are addressed in this paper.

1. How does integration of business application domain knowledge with technical knowledge during the software development process influence the following dimensions of development performance?
 - (1a) Software design effectiveness
 - (1b) Warranty and lifecycle defect density
 - (1c) Software development efficiency
2. What is the *relative* influence of knowledge integration on each of these dimensions of software development performance?

2. Prior research on knowledge management in software engineering

2.1. Knowledge in software development

Software development is a knowledge intensive process that involves the coordinated application of a variety of specialized knowledge in conceptualizing and designing a coherent software solution for a business problem. However, the interest in knowledge management as a mechanism for improving software practice is recent,

* Tel.: +1-404-727-6378; fax: +1-404-727-2053.

E-mail address: atiwana@bus.emory.edu (A. Tiwana).

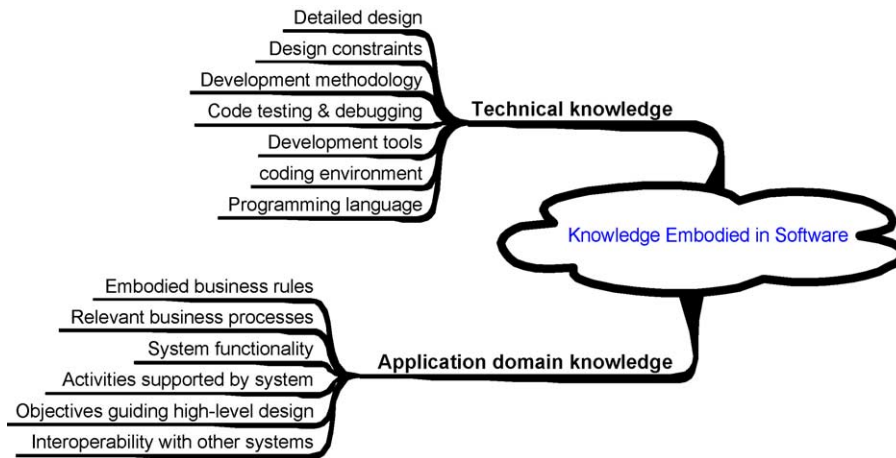


Fig. 1. Two key types of knowledge in software development.

much in response to the lackluster success rate of new software projects [12,15].¹

The role of knowledge in software development is widely recognized in several papers in the software engineering literature [1,6,14–16]. Rus and Lindvall [15] identify two types of knowledge that central to the software development process: (1) technical knowledge that is used to develop a system and (2) knowledge about the business application domain of a system. Technical knowledge refers to knowledge about design (e.g. design patterns, heuristics, best practices, technical constraints, and estimation models), programming (e.g. programming languages and development tools), and software processes (e.g. methodology, code testing and debugging procedures). Business application domain knowledge refers to knowledge about the customer's business processes, business rules, activities, stakeholder needs, and the customer's business objectives for the software. These are summarized in Fig. 1.

Although these two types of knowledge might be available to the software development team, it is necessary to integrate them in the design of the software. Such knowledge integration, we argue, is critical for the resulting system to meet its customer's/end-users' needs.

2.2. Knowledge integration in software development

A typical software development problem has multiple requirements, whose interactions and equivocality make them harder to satisfy. With growing complexity, even requirements that appear to be straightforward become too complex to grasp intuitively and early in the requirements elicitation process. Moreover, the complexity and tacitness of some requirements can make it difficult for the customer

to convey them precisely enough for the development team to formulate design specifications.

Achieving a good fit between the software design and its business objectives, therefore, requires organizing and integrating the specialized expertise, skills, and perspectives of various project stakeholders into an appropriate, coherent, and practical solution [7,12,13,15,16]. The two types of knowledge summarized in Fig. 1 must, therefore, be *coherently integrated and embodied in the design of software* for it to meet the customer's needs. Knowledge about customer needs must be embodied not just in the conceptual design, functionality, and features of the system but also in intermediate design artifacts such as contracts, development plans, requirements, and specifications [2,14,16]. We define this process of combining dispersed business domain and technical knowledge and embodying it in the design as *knowledge integration*.² It is essential that the two be integrated (see Fig. 2 for an illustration).

The actual design of the software is based on technical knowledge about programming languages, methodologies, and software architecture applied to the requirements expressed by or inferred from the customer organization. It is through the integration of business application domain knowledge and technical knowledge that they are converted into declarative knowledge—facts and properties about objects, persons, events, and relationships—that can be embodied in the software's design [14]. Therefore, knowledge about the problem domain becomes amenable to use in the development process through the process of knowledge integration.

¹ Of the \$2.5 trillion spent on IT during 1997–2001, nearly \$1 trillion was wagered on unsuccessful projects [10]. One in four software projects are canceled annually at a cost of \$67 billion. Cost overruns account for another \$21 billion in the United States alone.

² The importance of knowledge integration has implicitly been long recognized in the software engineering literature because many software engineering concepts such as information hiding, objects, patterns, functions and procedures, and modularity aim to guide or ease the processing of knowledge [14,17]. However, no prior study has empirically examined its effects.

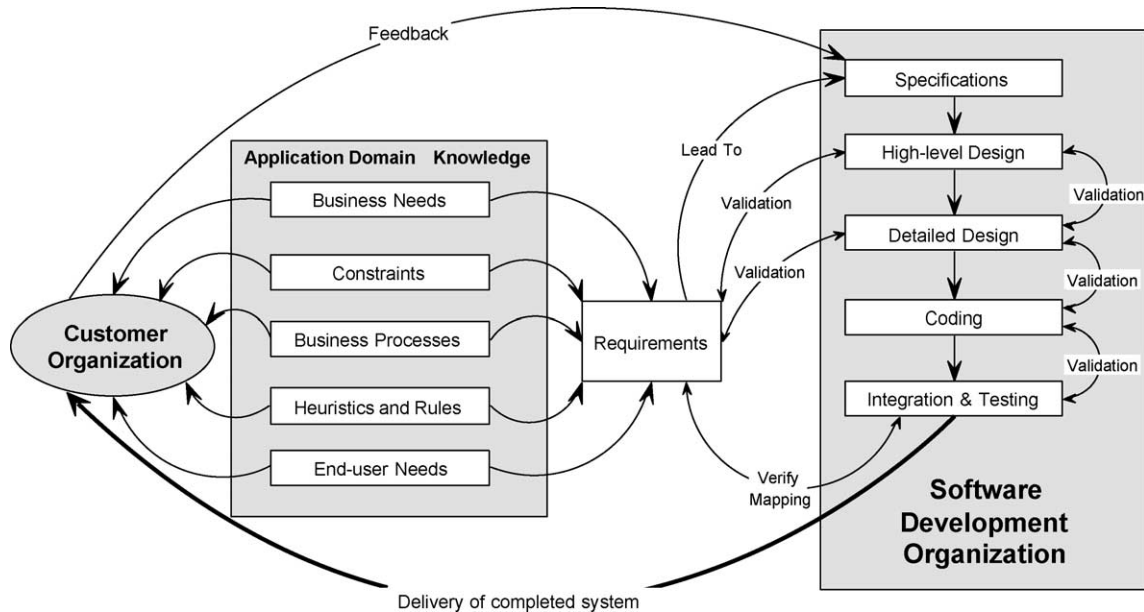


Fig. 2. Knowledge integration in a software development project.

The benefits of such integration extend beyond conceptualizing an effective preliminary design. As new information about evolving customer needs or changing project priorities surfaces during development, it is more likely to be reflected into the software design if high levels of knowledge integration are occurring [3]. Similarly, a recent case study of software development practices at NASA has suggested that coordination of effort during the development process reduces system costs [11]. However, the actual influence of knowledge integration on software development performance has never rigorously been empirically tested. Most evidence to date has either been anecdotal or based on case studies. In this study, we empirically assess how knowledge integration influences three key dimensions of software development performance: design effectiveness, defect density, and development efficiency.

We focus on these three variables because they are the preoccupation of both software development organizations (who seek to develop good products at the lowest possible cost) and the end-users or customers who generally seek to acquire the best possible solutions to their business problems at a reasonable cost. Design effectiveness captures the variable that determines whether the delivered system meets the needs of the customer organization. Defect density concerns both customers and software development organizations since it captures the number of defects that remain in the software per thousand lines of code. Efficiency of the development process concerns primarily the software development organization and it captures the effectiveness with which it can use its resources and skills to develop a system within budget.

3. Methodology

The field study involved 232 customer application development projects in 232 different software development organizations in Russia, Ireland, and India for external customer organizations, all of which were American companies. Data on each project were collected from two sources—the software development organization and the customer organization for which the software was developed. Through the three largest global software consortia in Russia (Russian National Software Development Alliance), Ireland (Irish Investment and Development Agency (IDA)), and India (National Association of Software and Service Companies (NASSCOM)) 818 potential participant software development organizations were identified. Only companies that had significant software development partnerships with US firms were chosen for this study. Throughout the study, recently proposed guidelines for empirical research on software engineering were closely followed [8].

3.1. Expert interviews and questionnaire development

The questionnaire used existing measurement scales where possible. A new measure was developed for knowledge integration based on detailed interviews with 19 software project managers (seven in Russia, six in the US, two in Ireland, and four in India) and seven academic experts. This measure was developed based on interviews with managers to identify ways in which integration of knowledge across the client–vendor boundary could be observed. Following this preliminary field work, a pool of scale items was generated. This pool was refined through

pilot testing and further rounds of feedback. The final measure utilized five questionnaire items that were all reflective scales [5]. Reflective scales refer to measures that tap into an unobservable concept (knowledge integration) using measurement items that are *caused* by the existence of that construct. An alternative approach is to use formative measures, i.e. where the measurement items *cause* the latent construct. The former approach was used because of existing scale validation procedures such as inter-item correlation can be used and no such reliability tests exist for formative scales.

Responses were measured using multi-item seven-point Likert scales and using objective data where possible. Five questionnaire items were used for each Likert scale construct to ensure reliable measurement. Principal components analysis with orthogonal rotation was used to confirm the factor loading patterns for each construct. All scales exhibited unidimensionality, discriminant validity, and reliably captured the underlying theoretical construct. The questionnaire items, Cronbach alphas, and factor loadings for each construct are summarized below.

1. *Knowledge integration* (Chronbach $\alpha = 0.74$) was measured using five questionnaire items on a seven-point Likert scale. These questions were answered by the software development organization. Each of these items was reflective in design, i.e. if higher levels of knowledge integration had occurred in the given project, they were designed to elicit a higher response on the response scale. A Chronbach Alpha value of 0.74 suggests that they tapped into a convergently valid unobservable construct [4]. The face validity of the scale items was assessed through an iterative scale development approach involving project managers from 19 organizations as described earlier. This ensured that the scale items reliability and validly tapped into the domain of the knowledge integration construct. The scale anchor was ‘For this project, please indicate the extent to which the following statements describe your company’s working relationship with this customer.’ The following items were used.

	Factor loading
(a) We applied our expertise in innovative ways	0.680
(b) We carefully made decisions to maximize overall project outcomes	0.636
(c) We leveraged the customer’s knowledge in many functional areas	0.695
(d) Many creative ideas came from combining our unique perspectives	0.768
(e) We developed a clear understanding of how each business function should be coordinated	0.600

2. *Design effectiveness* ($\alpha = 0.91$) was measured using five questionnaire items on a seven-point Likert scale answered by the customer organization. The scale anchor was ‘Compared to other IT projects completed by your company, how would you characterize this project’s outcomes?’ The following items were used.

	Factor loading
(a) System reliability	0.722
(b) Implementation of functionality	0.769
(c) Meeting project objectives	0.788
(d) Meeting functional requirements	0.800
(e) Overall fit with customer needs	0.813

3. *Defect density* was measured in terms of the total number of defects discovered in the software in its installation and acceptance testing stages (‘warranty defects’). The lower the defect count in these stages, the *higher* is the *design quality*.

4. *Development efficiency* was measured as the extent to which the project was completed in its allocated budget. This variable was measured inversely as the percentage of cost overruns encountered in the project.

3.2. Data collection

Data for the study were collected from key project managers in the software development organizations and their customer-side liaison managers through a field survey. The key informant technique, which relies on a highly knowledgeable individual from each organization was used [9]. Software development organizations in India, Ireland, and Russia were chosen for the study because of the large variance in software development practices, and therefore the large variance in knowledge integration that was likely (the data confirm this expectation). Alternatively, the phenomenon of knowledge integration could have been studied within the IT departments of organizations. However, the effects of knowledge integration were more likely to be observable in a field setting involving inter-organizational software development projects. The data collection was done in two phases, which were preceded by the questionnaire development interviews discussed earlier (see Fig. 3).

In the first phase, the software development organization questionnaire was forwarded to the lead manager of each project in each software development organization. We also separately collected objective archival data for each project (defects recorded at each stage of the development process and the percentage by which the project exceeded the original budget). Following this stage, we asked the project manager to forward a shorter survey to the primary liaison in the American customer organization. The purpose of this step was to collect data on the independent and dependent variables from

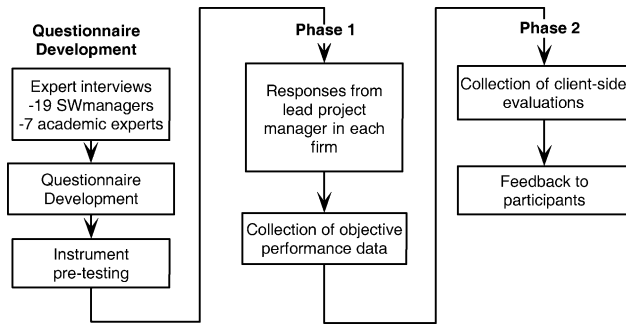


Fig. 3. Research methodology and data collection.

different respondents in order to mitigate the threat of mono-methods bias. Mono methods bias refers to bias in perceptual evaluations that results from having the same respondent evaluate both the dependent variables (performance) and the predictor variable (knowledge integration) in the same survey. Only projects for which complete matched-pair datasets were collected are included in the final analysis.

The response rates across the sample are summarized in Table 1. Various steps were taken to maximize response rates.³ Steps were also taken to ensure that non-response bias was not a persuasive threat to the validity of our findings.⁴ Given the sensitive nature of the data and the challenges associated with getting multiple assessments from both the customer and software development organizations, the overall response rate of 28.4% compares favorably to typical field studies involving managers.

4. Analysis and results

4.1. Model specification

The influence of knowledge integration on software development performance can be assessed by estimating three simple regression models for software development effectiveness (Eq. (1)), defect density (Eq. (2)), and development efficiency (Eq. (3)). The corresponding β_{ki} values in each model represent the standardized path coefficients that

³ Three steps were taken to maximize response rates. First, we promised complete confidentiality of responses and offered a summary of the key findings of the study customized to each participating company. Second, we send a pre-notification letter to all firms in the Russian and Irish cluster, and to the Indian ones for which mailing addresses could be obtained. Then we simultaneously sent the survey via email and in hard copy format with international return envelopes. Three reminders were sent via fax, telephone calls, and email. Third, preliminary comparative results and a token gift were sent to each responding firm along with the request for a customer-side evaluation.

⁴ To test whether the non-responding organizations were biasing our sample, we used follow-up calls to nine randomly selected non-responding software developments (three each in Ireland, Russia, and India). Most non-participating organizations declined to participate for lack of time, concerns about sensitivity of the data, or because they were no longer in the custom development business, thereby suggesting that non-response bias is not a persuasive threat to our findings.

Table 1
Response rate

	Russia	Ireland	India	Full sample
Valid initial sample	176	183	459	818
Respondents	59	54	119	232
Response rate	33.5%	29.5%	25.9%	28.4%

indicate the relative influence on each of these dimensions of software performance. The statistical significance of the betas indicates the robustness of hypothesized effects

$$\text{Design_effectiveness} = \beta_0 + \beta_{ki_effectiveness} * \text{Know_integ} + \varepsilon_1 \quad (1)$$

$$\text{Defect_density} = \beta_0 + \beta_{ki_defect} * \text{Know_integ} + \varepsilon_2 \quad (2)$$

$$\text{Design_efficiency} = \beta_0 + \beta_{ki_efficiency} * \text{Know_integ} + \varepsilon_3 \quad (3)$$

$\varepsilon_1 - \varepsilon_3$ represent the error terms.

Eqs. (1)–(3) were estimated using the field data on each of the 232 projects. Since the objective of this study is to empirically assess the robustness and degree to which knowledge integration influences each dimension of SD performance (rather than build a comprehensive predictive model), it is not necessary to complicate these equations further by introducing additional predictors of performance.

4.1.1. Descriptive statistics

The average complexity of the projects was 2241 function points (FPs). The duration of the projects ranged from one month to four years, with an average duration of about 11 months (SD 9.6 months). On average, the projects in the study had exceeded budget and schedule estimates by 15.7% (SD 16.1%) and 18.9% (SD 19.7%).

4.2. Model estimation

Three regression models corresponding to Eqs. (1)–(3) were estimated using the field data on all 232 projects. In our discussion, we use an alpha level of $P = 0.05$ to denote statistical significance. This corresponds to the 95% statistical confidence level. One-tailed T -tests were used because the hypothesized relationships are unidirectional. In our preliminary analysis, we found no statistically significant differences between the Russian, Irish, and Indian subsamples in our sample. The corresponding dummy variables were non-significant, suggesting that rational origin did not play an observable part in differentiating the performance of these firms. The projects were therefore, pooled in subsequent analyses. The path coefficients, their standardized error terms, and statistical significance were assessed using SPSS. The results are summarized in Table 2 and discussed next.

Table 2
Results

Equation	Performance dimension		Beta	Std. error	T-value	Statistical significance
1	Design effectiveness	$\beta_{ki_effectiveness}$	0.210	0.438	3.015	$P < 0.01$
2	Design defects	$\beta_{ki_defect\ density}$	-0.273	0.012	-3.997	$P < 0.001$
3	Cost overrun	$\beta_{ki_efficiency}$	-0.238	0.04	-2.276	$P < 0.05$

4.2.1. Effect on design effectiveness

Knowledge integration had a positive and statistically significant path coefficient on 0.210 on software design effectiveness (Eq. (1)). This provides evidence that higher levels of integration of business application domain and technical knowledge during the development process leads to improved fit between the delivered software and customer business needs. Recall that design effectiveness was evaluated by the customer organization, lending high levels of validity to this finding. This relationship is graphically shown in Fig. 4a, which shows a linear increase in design effectiveness as knowledge integration improves.

4.2.2. Effect on defect density

Estimation of Eq. (2) based on field data from 232 projects revealed that knowledge integration had a negative and statistically significant effect on the number of recorded defects in the installation and acceptance testing stages of development. The defect density metric is adjusted for the size of the project to allow comparisons across projects of different levels of complexity. We also examined the defect density patterns at each development stage—requirements, high-level design, detailed design, development/coding, installation and acceptance testing—and their relationship with knowledge integration (Fig. 5). This reveals an interesting pattern: the number of defects at each stage of development is lower in projects with higher levels of knowledge integration. This finding suggests that better knowledge integration *lowers the defect injection rate throughout the entire development trajectory*. Curiously, the inverse-U shape of the curves suggest that as knowledge

integration increases, the *recorded* defects initially rise and then fall. An explanation for this observation is that with improved knowledge integration, the ability of the software development organization to detect defects improves, as suggested by the higher number of recorded defects. On the other hand, these defects might otherwise simply go undetected.

4.2.3. Effect on software development efficiency

Estimation of Eq. (3) revealed a negative and statistically significant path coefficient between the percentage by which the project costs overran the allocated development costs. This implies that projects with higher levels of knowledge integration are completed with less deviation from the estimated budget, hence lead to more efficient utilization of the allocated resources. This relationship is shown in Fig. 4b, which reveals a linear downward trend in cost overruns as knowledge integration improves.

4.2.4. Comparative effects on development effectiveness, efficiency, and defect density

The path coefficients for the knowledge integration term in Eqs. (1)–(3) are standardized beta weights, which indicate the *comparative* effects of knowledge integration on development effectiveness, defect density, and development efficiency. The results in Table 2 suggest that the most pronounced influence of improved knowledge integration is on lowering the density of defects in the development process. The second dimension of performance that is most heavily influenced by it is the efficiency of the development process. This is followed closely by increased development

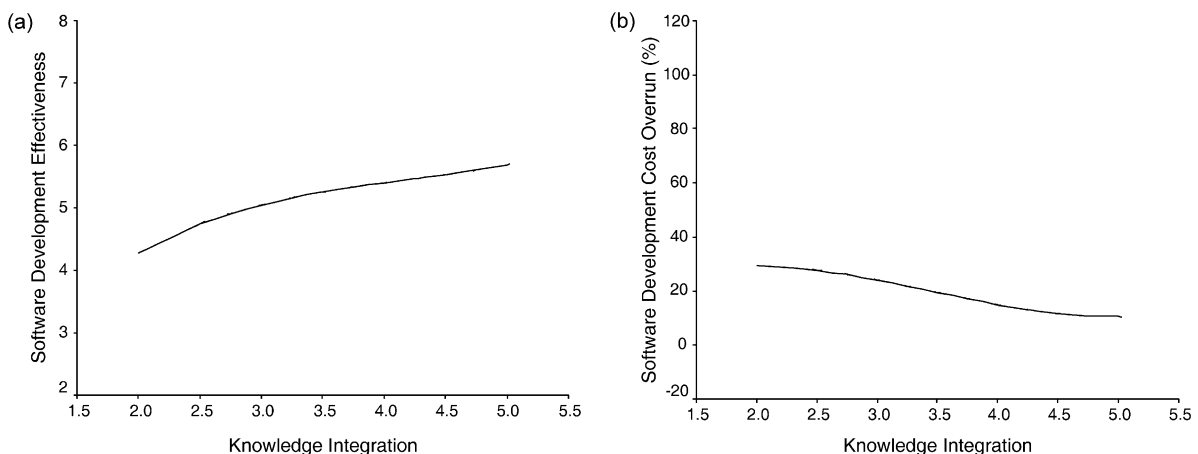


Fig. 4. Effect of knowledge integration on software development (a) effectiveness and (b) % cost overruns ($\propto 1/\text{efficiency}$).

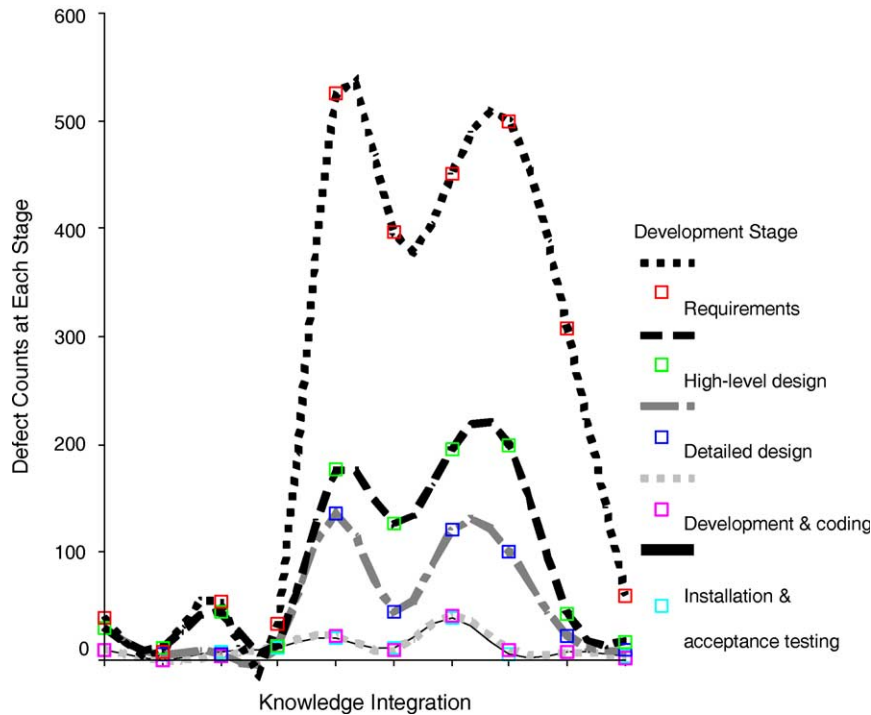


Fig. 5. Knowledge integration reduces defect density throughout the development trajectory.

effectiveness. Fig. 6 further shows that higher levels of knowledge integration are needed as the complexity of a software project (measured in person hours of development effort) grows.

5. Implications

Two key implications can be drawn from these findings. First, software development organizations must pay attention and resources to improve integration of their knowledge with the customer’s business application domain knowledge. Even if a customer possesses a clear mental model of a project’s objectives, unless it is integrated during the development process, it is unlikely that the resulting software will embody the customer’s actual needs. When knowledge about customer needs is complex and sticky, formal requirements are likely to capture only a subset of a project’s true requirements. In such cases, ongoing vendor–customer interaction throughout the development process can improve knowledge integration which in turn can help surface requirements that are not captured at the outset of the development process. This approach is advocated in iterative or evolutionary approaches for software development. The chronic challenges in software development—unmet user needs, requirements mismatches, and systems that fail to meet customer expectations—plausibly arise from inadequate integration of the business application domain knowledge with technical knowledge in the software development process. Since knowledge integration enhances software development effectiveness, software

quality, and efficiency, this facet of knowledge management can provide immediate impact in software development organizations considering knowledge management initiatives. Establishing a link between knowledge integration and software project outcomes is therefore, the key contribution of this study. Second, our findings also point to *why* development coordination tools such as requirements managers, architectural modeling tools, test case development tools, configuration managers, and defect and change request tracking tools enhance the software development process. Our findings suggest that it is so because they

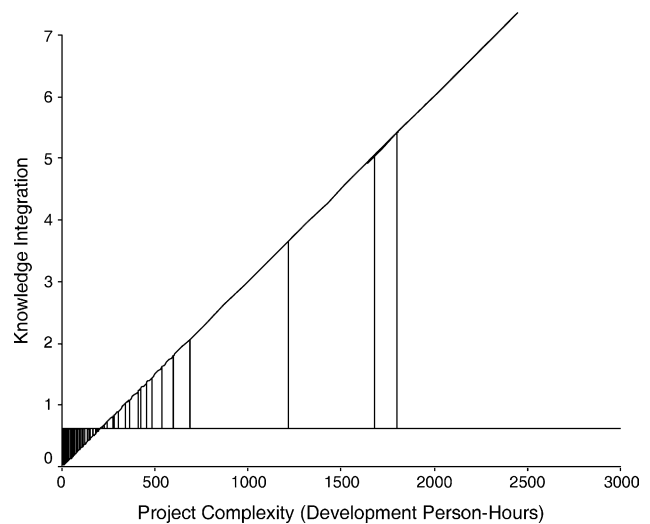


Fig. 6. Higher levels of knowledge integration are required in more complex projects.

enhance knowledge integration at the vendor–customer interface.

6. Summary

The objective of this research study was to assess empirically how robustly and to what extent integration of business application domain knowledge with technical knowledge during the software development process influences various dimensions of software development performance. Based on field data from 232 projects in 232 software development organizations and project evaluations by customer organizations, we found that knowledge integration has a statistically significant effect on software development performance. We found that knowledge integration—in the order of its effect size—lowers the density of warrant stage defects, lowers defect density throughout the development trajectory, improves software development efficiency, and increases design effectiveness. The key contributions of the study are: (1) validating the role of knowledge integration in the software development process for the first time and (2) assessing the relative extent to which each dimension of software development performance is influenced by it. Given the global nature of the dataset on which these results are based, these findings are readily generalizable to various organizational and national contexts.

Acknowledgements

The support of the Russian Development Alliance, Embassy of Ireland in Washington DC, Ireland Development Agency, and India's NASSCOM is gratefully acknowledged. The inputs of the following individuals in the various stages of the study are also acknowledged: Anandhi Bharadwaj, Ashley Bush, Mark Keil, Bala Ramesh, and Benn Konsynski, Dimitri Kroupski (Inform Link), Maria Dragan (Aplana Software), Pavel Kononov (JS RCOM), Masha Mezhburd (Tengry software), Andrew Cooper (Ectaco), Vladimir Lando (Krista Software), Michael Vasilenko (TESIS), Alexander Sergeev (EpsilonTech), Sergey Ushakov (INT), Sergei Make-donski (Market-Visio/EDC), Valentina Bukina (SPIRIT), Irina Lantuykhova (Avalon Tree), Andrei Teterevenkov (SITEch), Alexander Osipov (I-net Labs), Paul Bale (Sharptext Ltd), Seán Ryan (Aspen Grove Software),

Colm Doherty (XIAM Ltd), Deepak Kumar (SoftwareDi-oxide), Uday Rangaswamy (Global Software), U. Subramanian (HCC Infotech), Anwer Bagdadi (Lawkim), Sanjeev Kulkarni (Advent Software), Ajai Chowdhry (HCL Infosystems), and S. Chari (Onicra).

References

- [1] B. Adelson, E. Soloway, The role of domain experience in software design, *IEEE Transactions on Software Engineering* SE-11 (11) (1985) 1351–1360.
- [2] P. Armour, A case for a new business model: is software a product or a medium?, *Communications of the ACM* 43 (8) (2000) 19–22.
- [3] V. Basili, G. Caldiera, Improve software quality by reusing knowledge and experience, *Sloan Management Review* Fall (1995) 55–64.
- [4] D.T. Campbell, D.W. Fiske, Convergent and discriminant validation by the multitrait–multimethod matrix, *Psychological Bulletin* 56 (2) (1959) 81–105.
- [5] W. Chin, The partial least squares approach to structural equation modeling, in: G. Marcoulides (Ed.), *Modern Methods for Business Research*, Lawrence Erlbaum, Mahwah, NJ, 1998, pp. 295–336.
- [6] K. DeSouza, Barriers to effective use of knowledge management systems in software engineering, *Communications of the ACM* 46 (1) (2003) 99–101.
- [7] S. Faraj, L. Sproull, Coordinating expertise in software development teams, *Management Science* 46 (12) (2000) 1554–1568.
- [8] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, D. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, *IEEE Transactions on Software Engineering* 28 (8) (2002) 721–734.
- [9] N. Kumar, L.W. Stern, J.C. Anderson, Conducting interorganizational research using key informants, *Academy of Management Journal* 36 (6) (1993) 1633–1651.
- [10] C. Mann, Why is software so bad, *MIT Technology Review* Jul/Aug (2002) 33–38.
- [11] V.S. Mookerjee, R. Chiang, A. Dynamic, Coordination policy for software system construction, *IEEE Transactions on Software Engineering* 28 (6) (2002) 684–694.
- [12] B. Ramesh, Process knowledge management with traceability, *IEEE Software* May/Jun (2002) 50–55.
- [13] B. Ramesh, V. Dhar, Supporting systems development by capturing deliberations during requirements engineering, *IEEE Transactions on Software Engineering* 18 (1992) 498–510.
- [14] P. Robillard, The role of knowledge in software development, *Communications of the ACM* 42 (1) (1999) 87–92.
- [15] I. Rus, M. Lindvall, Knowledge management in software engineering, *IEEE Software* May/Jun (2002) 26–38.
- [16] D. Walz, J. Elam, B. Curtis, Inside a software design team: knowledge sharing, and integration, *Communications of the ACM* 36 (10) (1993) 63–77.
- [17] S.H. Zweben, S.H. Edwards, B.W. Weide, J.E. Hollingsworth, The effects of layering and encapsulation on software development cost and quality, *IEEE Transactions on Software Engineering [ISO]* 21 (3) (1995) 200–208.